Oracle8*i*

Java Tools Reference

Release 3 (8.1.7)

July 2000

Part No. A83727-01

Java Tools Reference, Release 3 (8.1.7)

Part No. A83727-01

Release 3 (8.1.7)

Primary Authors: Sheryl Maring

Contributors: Jason Minton, Dmitry Nizhegorodov, Wendy Liau, Sastry Malladi, John O’Duinn, Aniruddha Thakur, Jerry Schwarz, Steve Harris, Sam Chou

Contents

Send Us Your Comments

Oracle8*i* Java Tools Reference, Release 3 (8.1.7)

Part No. A83727-01

Oracle Corporation welcomes your comments and suggestions on the quality and usefulness of this publication. Your input is an important part of the information used for revision.

- Did you find any errors?
- Is the information clearly presented?
- Do you need more information? If so, where?
- Are the examples correct? Do you need more examples?
- What features did you like most about this manual?

If you find any errors or have any other suggestions for improvement, please indicate the chapter, section, and page number (if available). You can send comments to us in the following ways:

- Electronic mail — jpgcomnt@us.oracle.com
- FAX - 650-506-7225. Attn: Java Platform Group, Information Development Manager
- Postal service:
  Oracle Corporation
  Information Development Manager
  500 Oracle Parkway, Mailstop 4op978
  Redwood Shores, CA 94065
  USA

Please indicate if you would like a reply.

If you have problems with the software, please contact your local Oracle World Wide Support Center.

Preface

This reference contains the syntax and description for Oracle8*i* JServer command-line tools.

How This Reference is Organized

This book has the following two chapters:

- Chapter 1, "Tools" describes all of the tools that support Java development within Oracle8*i* version 8.1.7.
- Chapter 2, "Backwards Compatibility Tools" describes tools that have been deprecated for 8.1.7, but that still can be used for existing 8.1.6 Java development.

Notational Conventions

This guide follows these conventions:

Italic	Italic font denotes terms being defined for the first time, words being emphasized, error messages, and book titles.
`Courier`	Courier font denotes Java program names, file names, path names, and Internet addresses.

Java code examples follow these conventions:

`{ }`	Braces enclose a block of statements.
`//`	A double slash begins a single-line comment, which extends to the end of a line.
`/*   */`	A slash-asterisk and an asterisk-slash delimit a multi-line comment, which can span multiple lines.
`...`	An ellipsis shows that statements or clauses irrelevant to the discussion were left out.
`lower case`	Lower case is used for keywords and for one-word names of variables, methods, and packages.
`UPPER CASE`	Upper case is used for names of constants (static final variables) and for names of supplied classes that map to built-in SQL datatypes.
`Mixed Case`	Mixed case is used for names of classes and interfaces and for multi-word names of variables, methods, and packages. The names of classes and interfaces begin with an upper-case letter. In all multi-word names, the second and succeeding words also begin with an upper-case letter.

Your Comments Are Welcome

We appreciate your comments and suggestions. In fact, your opinions are the most important feedback we receive. We encourage you to use the Reader's Comment Form at the front of this book. You can also send comments to the following address:

Documentation Manager, Oracle8*i* Java Products Group
Oracle Corporation
500 Oracle Parkway
Redwood Shores, CA 94065
USA
email: jpgcomnt@us.oracle.com

1
Tools

This chapter describes the tools you use in the Oracle8*i* Java environment. You run these tools from a UNIX shell or the Windows NT DOS prompt.

Note: All names supplied within these tools are case sensitive. Thus, the schema, username, and password will not be uppercased.

The tools described in this chapter as follows:

- Schema Object Tools
- Session Namespace Tools
- Enterprise JavaBean Tools
- VisiBroker™ for Java Tools
- Native Compilation Tools
- Miscellaneous Tools

Schema Object Tools

Unlike a conventional Java virtual machine, which compiles and loads Java files, the Aurora Java virtual machine compiles and loads schema objects. The three kinds of Java schema objects are:

- *Java class schema objects*, which correspond to Java class files.
- *Java source schema objects*, which correspond to Java source files.
- *Java resource schema objects*, which correspond to Java resource files.

To make a class file runnable by the Aurora Java virtual machine, you use the `loadjava` tool to create a Java class schema object from the class file or the source file and load it into a schema. To make a resource file accessible to the Aurora Java virtual machine, you use `loadjava` to create and load a Java resource schema object from the resource file.

The `dropjava` tool does the reverse of the `loadjava` tool; it deletes schema objects that correspond to Java files. You should always use `dropjava` to delete a Java schema object that was created with `loadjava`; dropping by means of SQL DDL commands will not update auxiliary data maintained by `loadjava` and `dropjava`.

What and When to Load

You must load resource files with `loadjava`. If you create `.class` files outside the database with a conventional compiler, then you must load them with `loadjava`. The alternative to loading class files is to load source files and let the Oracle8*i* system compile and manage the resulting class schema objects. In the current Oracle8*i* release, most developers will find that compiling and debugging most of their code outside the database and then loading `.class` files to debug those files which must be tested inside the database, is the most productive approach. For a particular Java class, you can load either its `.class` file or its `.java` file, but not both.

`loadjava` accepts JAR files that contain either source and resource files or class and resource files (recall that you can load a class's source or its class file but not both). When you pass `loadjava` a JAR file or a ZIP file, `loadjava` opens the archive and loads its members individually; there is no JAR or ZIP schema object. A file whose content has not changed since the last time it was loaded is not re-loaded, therefore there is little performance penalty for loading JARs. Loading JAR files is the simplest and most foolproof way to use `loadjava`.

It is illegal for two schema objects in the same schema to define the same class. For example, suppose `a.java` defines class `x` and you want to move the definition of `x` to `b.java`. If `a.java` has already been loaded, then `loadjava` will reject an attempt to load `b.java` (which also defines `x`). Instead, do either of the following:

- Drop `a.java`, load `b.java` (which defines `x`), then load the new `a.java` (which does not define `x`).
- Load the new `a.java` (which does not define `x`), then load `b.java` (which defines `x`).

Resolution

All Java classes contain references to other classes. A conventional Java virtual machine searches for classes in the directories, ZIP files, and JARs named in the CLASSPATH. The Aurora Java virtual machine, by contrast, searches schemas for class schema objects. Each Oracle8*i* class has a *resolver spec*, which is the Oracle8*i* counterpart to the CLASSPATH. For a hypothetical class `alpha`, its resolver spec is a list of schemas to search for classes `alpha` uses. Notice that resolver specs are per-class, whereas in a classic Java virtual machine, CLASSPATH is global to all classes.

In addition to a resolver spec, each class schema object has a list of interclass reference bindings. Each reference list item contains a reference to another class, and one of the following:

- the name of the class schema object to invoke when class uses the reference
- a code indicating that the reference is unsatisfied; in other words, the referent schema object is not known

An Oracle8*i* facility called the *resolver* maintains reference lists. For each interclass reference in a class, the resolver searches the schemas specified by the class's resolver spec for a valid class schema object that satisfies the reference. If all references are resolved, the resolver marks the class *valid*. A class that has never been resolved, or has been resolved unsuccessfully, is marked *invalid*. A class that depends on a schema object that becomes invalid is also marked invalid at the same time; in other words, invalidation cascades upward from a class to the classes that use it and the classes that use them, and so on. When resolving a class that depends on an invalid class, the resolver first tries to resolve the dependency because it may be marked invalid only because it has never been resolved. The resolver does not re-resolve classes that are marked valid.

A class developer can direct `loadjava` to resolve classes, or can defer resolution until run time. (The resolver runs automatically when a class tries to load a class that is marked invalid.) It is best to resolve before run time to learn of missing classes early; unsuccessful resolution at run time produces a "class not found" exception. Furthermore, run-time resolution can fail for lack of database resources if the tree of classes is very large.

The `loadjava` has two resolution modes:

1. Load-and-resolve (`-resolve` option): Loads all classes you specify on the command line, marks them invalid, and then resolves them. Use this mode when initially loading classes that refer to each other, and in general when reloading isolated classes as well. By loading all classes and then resolving them, this mode avoids the error message that occurs if a class refers to a class that will be loaded later in the execution of the command.
2. Load-then-resolve (no `-resolve` option): Resolves each class when compiled at runtime.

Note: Like a Java compiler, `loadjava` resolves references to classes but not to resources; be sure to correctly load the resource files your classes need.

If you can, it is best to defer resolution until all classes have been loaded; this technique avoids the situation in which the resolver marks a class invalid because a class it uses has not yet been loaded.

Digest Table

The schema object digest table is an optimization that is usually invisible to developers. The digest table enables `loadjava` to skip files that have not changed since they were last loaded. This feature improves the performance of makefiles and scripts that invoke `loadjava` for collections of files, only some of which need to be re-loaded. A re-loaded archive file might also contain some files that have changed since they were last loaded and some that have not.

The `loadjava` tool detects unchanged files by maintaining a digest table in each schema. The digest table relates a file name to a *digest*, which is a shorthand representation of the file's content (a hash). Comparing digests computed for the same file at different times is a fast way to detect a change in the file's content—much faster than comparing every byte in the file. For each file it processes, `loadjava` computes a digest of the file's content and then looks up the file name in the digest table. If the digest table contains an entry for the file name that has the identical digest, then `loadjava` does not load the file because a corresponding schema object exists and is up to date. If you invoke `loadjava` with the `-verbose` option, then it will show you the results of its digest table lookups.

Normally, the digest table is invisible to developers because `loadjava` and `dropjava` keep it synchronized with schema object additions, changes, and deletions. For this reason, always use `dropjava` to delete a schema object that was created with `loadjava`, even if you know how to drop a schema object with DDL. If the digest table becomes corrupted (`loadjava` does not update a schema object whose file has changed), use `loadjava`'s `-force` option to bypass the digest table lookup.

Compilation

Loading a source file creates or updates a Java source schema object and invalidates the class schema object(s) previously derived from the source. (If the class schema objects don't exist, `loadjava` creates them.) `loadjava` invalidates the old class schema objects because they were not compiled from the newly loaded source. Compilation of a newly loaded source, called for instance A, is automatically triggered by any of the following conditions:

- The resolver, working on class B, finds that it refers to class A but class A is invalid.
- The compiler, compiling source B, finds that it refers to class A but A is invalid.
- The class loader, trying to load class A for execution, finds that it is invalid.

To force compilation when you load a source file, use `loadjava -resolve`.

The compiler writes error messages to the predefined `USER_ERRORS` view; `loadjava` retrieves and displays the messages produced by its compiler invocations. See the *Oracle8i Reference* for a description of this table.

The compiler recognizes compiler options. There are two ways to specify options to the compiler. If you run `loadjava` with the `-resolve` option (which may trigger compilation), you can specify compiler options on the command line.

You can additionally specify persistent compiler options in a per-schema database table called `JAVA$OPTIONS` which you create as described shortly. You can use the `JAVA$OPTIONS` table for default compiler options, which you can override selectively with a `loadjava` command-line option.

Note: A command-line option both overrides and clears the matching entry in the `JAVA$OPTIONS` table.

A JAVA$OPTIONS row contains the names of source schema objects to which an option setting applies; you can use multiple rows to set the options differently for different source schema objects. The compiler looks up options in the JAVA$OPTIONS table when it has been invoked without a command line (that is, by the class loader), or when the command line does not specify an option. When compiling a source schema object for which there is neither a JAVA$OPTIONS entry nor a command-line value for an option, the compiler assumes a default value as follows:

- encoding = latin1
- online = true: see the *Oracle8i SQLJ Developer's Guide and Reference* for a description of this option, which only applies to Java sources that contain SQLJ constructs.

You can set JAVA$OPTIONS entries by means of the following functions and procedures, which are defined in the database package DBMS_JAVA:

- `PROCEDURE set_compiler_option(name VARCHAR2, option VARCHAR2, value VARCHAR2);`
- `FUNCTION get_compiler_option(name VARCHAR2, option VARCHAR2) RETURNS VARCHAR2;`
- `PROCEDURE reset_compiler_option(name VARCHAR2, option VARCHAR2);`

The name parameter is a Java package name, or a fully qualified class name, or the empty string. When the compiler searches the JAVA$OPTIONS table for the options to use for compiling a Java source schema object, it uses the row whose name most closely matches the schema object's fully qualified class name. A name whose value is the empty string matches any schema object name.

The option parameter is either 'online' or 'encoding'. For the values you can specify for these options, see the *Oracle8i SQLJ Developer's Guide and Reference*.

A schema does not initially have a JAVA$OPTIONS table. To create a JAVA$OPTIONS table, use the DBMS_JAVA package's java.set_compiler_option procedure to set a value; the procedure will create the table if it does not exist. Specify parameters in single quotes. For example:

```
SQL> execute dbms_java.set_compiler_option('x.y', 'online', 'false');
```

Table 1–1 represents a hypothetical JAVA$OPTIONS database table. Because the table has no entry for the encoding option, the compiler will use the default or the value specified on the command line. The `online` options shown in the table match schema object names as follows:

- The name `a.b.c.d` matches class and package names beginning with `a.b.c.d`; they will be compiled with `online` = `true`.
- The name `a.b` matches class and package names beginning with `a.b` but not `a.b.c.d`; they will be compiled with `online` = `false`.
- All other packages and classes will match the empty string entry and will be compiled with `online` = `true`.

Table 1–1 Example JAVA$OPTIONS Table and Matching Examples

JAVA$OPTIONS Entries			**Match Examples**
Name	**Option**	**Value**	
`a.b.c.d`	`online`	`true`	`a.b.c.d, a.b.c.d.e`
`a.b`	`online`	`false`	`a.b, a.b.c.x`
(empty string)	`online`	`true`	`a.c, x.y`

loadjava

The `loadjava` tool creates schema objects from files and loads them into a schema. Schema objects can be created from Java source, class, and data files. `loadjava` can also create schema objects from SQLJ files; the *Oracle8i SQLJ Developer's Guide and Reference* describes how to use `loadjava` with SQLJ.

You must have the following SQL database privileges to load classes:

- `CREATE PROCEDURE` and `CREATE TABLE` privileges to load into your schema.
- `CREATE ANY PROCEDURE` and `CREATE ANY TABLE` privileges to load into another schema.
- oracle.aurora.security.JServerPermission.loadLibraryInClass.<*classname*>. See the "Database Contents and JVM Security" section in Chapter 5 of the *Oracle8i Java Developer's Guide* for more information.

You can execute the `loadjava` tool either through the command line (as described below) or through the `loadjava` method contained within the `DBMS_JAVA` class. To execute within your Java application, do the following:

```
call dbms_java.loadjava('... options...');
```

where the options are the same as specified below. Each option should be separated by a blank. You should not separate the options with a comma. The only exception for this is the `-resolver` option, which contains blanks. For `-resolver`, you should specify all other options first, a comma, then the `-resolver` option with its definition. You should not specify the following options as they relate to the database connection for the `loadjava` command-line tool: `-thin`, `-oci8`, `-user`, `-password`. The output is directed to stderr. Set serveroutput on and call `dbms_java.set_output` as appropriate.

Note: The `loadjava` tool is located in the `bin` subdirectory under $ORACLE_HOME.

Syntax

```
loadjava {-user | -u} <user>/<password>[@<database>] [options]
<file>.java | <file>.class | <file>.jar | <file>.zip |
<file>.sqlj | <resourcefile> ...
  [-debug]
  [-d | -definer]
  [-e | -encoding <encoding_scheme>]
  [-f | -force]
  [-g | -grant <user> [, <user>]...]
  [-help]
  [-nohelp]
  [-o | -oci8]
  [ -order ]
  [-noverify]
  [-r | -resolve]
  [-R | -resolver "resolver_spec"]
  [-S | -schema <schema>]
  [ -stdout ]
  [-s | -synonym]
  [-tableschema <schema>]
  [-t | -thin]
  [-v | -verbose]
```

Argument Summary

Table 1–2 summarizes the `loadjava` arguments. If you execute `loadjava` multiple times specifying the same files and different options, the options specified in the most recent invocation hold. There are two exceptions:

1. If `loadjava` does not load a file because it matches a digest table entry, most options on the command line have no effect on the schema object. The

exceptions are `-grant`, and `-resolve`, which are always obeyed. Use the `-force` option to direct `loadjava` to skip the digest table lookup.

2. The `-grant` option is cumulative; every user specified in every `loadjava` invocation for a given class in a given schema has the EXECUTE privilege. You cannot grant to a role; you can only grant to specified schemas or users.

Table 1–2 loadjava Argument Summary

Argument	Description
`<filenames>`	You can specify any number and combination of `.java`, `.class`, `.sqlj`, `.ser`, `.jar .zip`, and resource file name arguments in any order.
`-debug`	Turns on SQL logging and is equivalent to `javac -g`.
`-definer`	By default, class schema objects run with the privileges of their invoker. This option confers definer (the developer who invokes `loadjava`) privileges upon classes instead. (This option is conceptually similar to the UNIX `setuid` facility.)
`-encoding`	Identifies the source file encoding for the compiler, overriding the matching value, if any, in the `JAVA$OPTIONS` table. Values are the same as for the `javac -encoding` option. If you do not specify an encoding on the command line or in a `JAVA$OPTIONS` table, the encoding is assumed to be `latin1`. The `-encoding` option is relevant only when loading a source file.
`-force`	Forces files to be loaded even if they match digest table entries.
`-grant`	Grants the EXECUTE privilege on loaded classes to the listed users. (To call the methods of a class, users must have the EXECUTE privilege.) Any number and combination of user names can be specified, separated by commas but not spaces (`-grant Bob,Betty` not `-grant Bob, Betty`). Note: `-grant` is a "cumulative" option; users are added to the list of those with the EXECUTE privilege. To remove privileges, either drop and reload the schema object with the desired privileges or change the privileges with the SQL REVOKE command. Also, you cannot grant to a role. All grants must be explicit in granting to specific users. To grant the EXECUTE privilege on an object in someone else's schema requires that the original CREATE PROCEDURE privilege was granted with WITH GRANT options. **Note**: You must uppercase the schema name.
`-help`	Prints the usage message on how to use the loadjava tool and its options.

Table 1–2 loadjava Argument Summary (Cont.)

Argument	Description
`-nohelp`	Suppresses the usage message that is given if either no option is specified or if the -help option is specified.
`-noverify`	Causes the classes to be loaded without bytecode verification. You must be granted oracle.aurora.security.JServerPermission(Verifier) to execute this option. In addition, this option must be used in conjunction with -r.
`-oci8`	Directs `loadjava` to communicate with the database using the OCI JDBC driver. `-oci8` and `-thin` are mutually exclusive; if neither is specified `-oci8` is used by default. Choosing `-oci8` implies the syntax of the `-user` value. You do not need to provide the URL.
`-order`	Directs `loadjava` load the classes in an order that facilitates resolution of those classes. Classes are loaded in a manner where any dependent class is loaded before the class that includes it as a dependency.
`-resolve`	Compiles (if necessary) and resolves external references in classes after all classes on the command line have been loaded. If you do not specify -resolve, `loadjava` loads files but does not compile or resolve them.
`-resolver`	Specifies an explicit resolver spec, which is bound to the newly loaded classes. If -resolver is not specified, the default resolver spec, which includes current user's schema and PUBLIC, is used. See "resolver" on page 1-13 for details.
`-schema`	Designates the schema where schema objects are created. If not specified, the logon schema is used. To create a schema object in a schema that is not your own, you must have the CREATE PROCEDURE or CREATE ANY PROCEDURE privilege. You must have CREATE TABLE or CREATE ANY TABLE privilege. Finally, you must have the JServerPermission.loadLibraryInClass for the class.
`-stdout`	Causes the output to be directed to `stdout`, rather than to `stderr`.
`-synonym`	Creates a PUBLIC synonym for loaded classes making them accessible outside the schema into which they are loaded. To specify this option, you must have the CREATE PUBLIC SYNONYM privilege. If `-synonym` is specified for source files, classes compiled from the source files are treated as if they had been loaded with `-synonym`.

Table 1–2 loadjava Argument Summary (Cont.)

Argument	Description
`-tableschema <schema>`	Creates the loadjava internal tables within this specified schema, rather than in the Java file destination schema.
`-thin`	Directs `loadjava` to communicate with the database using the thin JDBC driver. `-oci8` and `-thin` are mutually exclusive; if neither is specified, then `-oci8` is used by default. Choosing `-thin` implies the syntax of the `-user` value. You do need to specify the appropriate URL through the `-user` option.
`-user`	Specifies a user, password, and database connect string; the files will be loaded into this database instance. The argument has the form `<username>/<password>[@<database>]`.
-verbose	Directs loadjava to emit detailed status messages while running. Use `-verbose` to learn when `loadjava` does not load a file because it matches a digest table entry.

Argument Details

This section describes the details of `loadjava` arguments whose behavior is more complex than the summary descriptions contained in Table 1–2.

File Names

You can specify as many `.class`, `.java`, `.sqlj`, `.jar`, `.zip`, and resource files as you like, in any order. If you specify a JAR or ZIP file, then `loadjava` processes the files in the JAR or ZIP; there is no JAR or ZIP schema object. If a JAR or ZIP contains a JAR or ZIP, `loadjava` does not process them.

The best way to load files is to put them in a JAR or ZIP and then load the archive. Loading archives avoids the resource schema object naming complications described later in this section. If you have a JAR or ZIP that works with the JDK, then you can be sure that loading it with `loadjava` will also work, without having to learn anything about resource schema object naming.

Schema object names are slightly different from file names, and `loadjava` names different types of schema objects differently. Because class files are self-identifying (they contain their names), `loadjava`’s mapping of class file names to schema object names is invisible to developers. Source file name mapping is also invisible to developers; `loadjava` gives the schema object the fully qualified name of the first class defined in the file. JAR and ZIP files also contain the names of their files; however, resource files are not self identifying. `loadjava` generates Java resource schema object names from the *literal* names you supply as arguments (or the literal names in a JAR or ZIP file). Because running classes use resource schema objects, it is important that you specify resource file names correctly on the command line, and the correct specification is not always intuitive. The surefire way to load individual resource files correctly is:

Run `loadjava` from the top of the package tree and specify resource file names relative to that directory. (The "top of the package tree" is the directory you would name in a Java CLASSPATH list.)

If you do not want to follow this rule, observe the details of resource file naming that follow. When you load a resource file, `loadjava` generates the resource schema object name from the resource file name *as literally specified on the command line*. Suppose, for example you type:

```
% cd /home/scott/javastuff
% loadjava options alpha/beta/x.properties
% loadjava options /home/scott/javastuff/alpha/beta/x.properties
```

Although you have specified the same file with a relative and an absolute path name, `loadjava` creates *two* schema objects, one called `alpha/beta/x.properties`, the other `ROOT/home/scott/javastuff/alpha/beta/x.properties`. (`loadjava` prepends `ROOT` because schema object names cannot begin with the "/" character; however, that is an implementation detail that is unimportant to developers.) The important point is that a resource schema object's name is generated from the file name *as entered*.

Classes can refer to resource files relatively (for example, `b.properties`) or absolutely (for example, `/a/b.properties`). To ensure that `loadjava` and the class loader use the same name for a schema object, follow this rule when loading resource files:

Enter the name on the command line that the class passes to `getResource()` or `getResourceAsString()`.

Instead of remembering whether classes use relative or absolute resource names and changing directories so that you can enter the correct name on the command line, you can load resource files in a JAR as follows:

```
% cd /home/scott/javastuff
% jar -cf alpharesources.jar alpha/*.properties
% loadjava options alpharesources.jar
```

Or, to simplify further, put both the class and resource files in a JAR, which makes the following invocations equivalent:

```
% loadjava options alpha.jar
% loadjava options /home/scott/javastuff/alpha.jar
```

The two `loadjava` commands in this example make the point that you can use any pathname to load the contents of a JAR file. Note as well that even if you did execute the redundant commands shown above, `loadjava` would realize from the digest table that it did not need to load the files twice. That means that re-loading JAR files is not as time-consuming as it might seem even when few files have changed between `loadjava` invocations.

definer

`{-definer | -d}`

The `-definer` option is identical to definer's rights in stored procedures and is conceptually similar to the UNIX `setuid` facility; however, whereas `setuid` applies to a complete program, you can apply `-definer` class by class. Moreover, different definers may have different privileges. Because an application may consist of many classes, you must apply `-definer` with care to achieve the results desired, namely classes that run with the privileges they need but no more. For more information on definer's rights, see the *Oracle8i Java Stored Procedures Developer's Guide*.

noverify

`{-noverify]`

Causes the classes to be loaded without bytecode verification. You must be granted `oracle.aurora.security.JServerPermission(Verifier)` to execute this option. In addition, this option must be used in conjunction with `-r`.

The verifier ensures that incorrectly formed Java binaries cannot be loaded for execution in the server. If you know that the JAR or classes you are loading are valid, use of this option will speed up the `loadjava` process. Some JServer-specific optimizations for interpreted performance are put in place during the verification process. Thus, interpreted performance of your application may be adversely affected by using this option.

resolve

`{-resolve | -r}`

Use `-resolve` to force `loadjava` to compile (if necessary) and resolve a class that has previously been loaded. It is not necessary to specify `-force` because resolution is performed after, and independently of, loading.

resolver

`{-resolver | -R} "resolver spec"`

This option associates an explicit resolver spec with the class schema objects that `loadjava` creates or replaces.

A resolver spec consists of one or more items, each of which consists of a *name spec* and a *schema spec* expressed in the following syntax:

```
"((name_spec schema_spec) [(name_spec schema_spec)] ...)"
```

- A name spec is similar to a name in a Java `import` statement. It can be a fully qualified Java class name, or a package name whose final element is the wildcard character "`*`", or (unlike an imported package name) simply the wildcard character "`*`"; however, the elements of a name spec must be separated by "`/`" characters, not periods. For example, the name spec `a/b/*` matches all classes whose names begin with `a.b.`. The special name `*` matches all class names.
- A schema spec can be a schema name or the wildcard character "`-`". The wildcard does not identify a schema but directs the resolve operation to not mark a class invalid because a reference to a matching name cannot be resolved. (Without a "`-`" wildcard in a resolver spec, an unresolved reference in the class makes the class invalid and produces an error message.) Use a "`-`" wildcard when you must test a class that refers to a class you cannot or do not want to load; for example, GUI classes that a class refers to but does not call because when run in the server there is no GUI.

The resolution operation interprets a resolver spec item as follows:

When looking for a schema object whose name matches the name spec, look in the schema named by the partner schema spec.

The resolution operation searches schemas in the order in which the resolver spec lists them. For example,

```
-resolver '((* SCOTT) (* PUBLIC))'
```

means the following:

Search for any reference first in SCOTT and then in PUBLIC. If a reference is not resolved, then mark the referring class invalid and display an error message; in other words, call attention to missing classes.

The following example:

```
-resolver "((* SCOTT) (* PUBLIC) (my/gui/* -))"
```

means the following:

Search for any reference first in SCOTT and then in PUBLIC. If the reference is not found, and is to a class in the package my.gui then mark the referring class valid, and do not display an error; in other words, ignore missing classes in this package.

If the reference is not found and is not to a class in my.gui, then mark the referring class invalid and produce an error message.

user

`{-user | -u} <user>/<password>[@<database>]`

By default, `loadjava` loads into the login schema specified by the `-user` option. Use the `-schema` option to specify a different schema to load into. This does not involve a login into that schema, but does require that you have sufficient permissions to alter it.

The permissible forms of `@<database>` depend on whether you specify `-oci8` or `-thin`; `-oci8` is the default.

- `-oci8`: `@<database>` is optional; if you do not specify, `loadjava` uses the user's default database. If specified, `<database>` can be a TNS name or a Net8 name-value list.
- `-thin`: `@<database>` is required. The format is `<host>:<lport>:<SID>`.
  - `<host>` is the name of the machine running the database.
  - `<lport>` is the listener port that has been configured to listen for Net8 connections; in a default installation, it is 5521.
  - `<SID>` is the database instance identifier; in a default installation it is `ORCL`.

Here are examples of `loadjava` commands:

- Connect to the default database with the default oci8 driver, load the files in a JAR into the TEST schema, then resolve them.

  ```
  loadjava -u joe/shmoe -resolve -schema TEST ServerObjects.jar
  ```

- Connect with the thin driver, load a class and a resource file, and resolve each class:

  ```
  loadjava -thin -u SCOTT/TIGER@dbhost:5521:orcl \
    -resolve alpha.class beta.props
  ```

- Add Betty and Bob to the users who can execute `alpha.class`:

  ```
  loadjava -thin -schema test -u SCOTT/TIGER@localhost:5521:orcl \
    -grant BETTY,BOB alpha.class
  ```

dropjava

The dropjava tool is the converse of loadjava. It transforms command-line file names and JAR or ZIP file contents to schema object names, then drops the schema objects and deletes their corresponding digest table rows. You can enter .java, .class, .sqlj, .ser, .zip, .jar, and resource file names on the command line in any order.

Alternatively, you can specify a schema object name (full name, not short name) directly to dropjava. A command-line argument that does not end in .jar, .zip, .class, .java, or .sqlj is presumed to be a schema object name. If you specify a schema object name that applies to multiple schema objects (such as a source schema object Foo and a class schema object Foo), all will be removed.

Dropping a class invalidates classes that depend on it, recursively cascading upwards. Dropping a source drops classes derived from it.

Note: You must remove Java schema objects in the same way that you first loaded them. If you load a .sqlj source file and translate it in the server, you must run dropjava on the same source file. If you translate on a client and load classes and resources directly, run dropjava on the same classes and resources.

You can execute the dropjava tool either through the command line (as described below) or through the dropjava method contained within the DBMS_JAVA class. To execute within your Java application, do the following:

```
call dbms_java.dropjava('... options...');
```

where the options are the same as specified below. Each option should be separated by a blank. You should not separate the options with a comma. The only exception for this is the -user option. The connection is always made to the current session, so you cannot specify another username through the -user option.

For -resolver, you should specify all other options first, a comma, then the -resolver option with its definition. You should not specify the following options as they relate to the database connection for the loadjava command-line tool: -thin, -oci8, -user, -password. The output is directed to stderr. Set serveroutput on and call dbms_java.set_output as appropriate.

Syntax

```
dropjava [options] {<file>.java | <file>.class | file.sqlj |
<file>.jar | <file.zip> | <resourcefile>} ...
  -u | -user <user>/<password>[@<database>]
  [-o | -oci8]
  [-S | -schema <schema>]
  [ -stdout ]
  [-s | -synonym]
  [-t | -thin]
  [-v | -verbose]
```

Argument Summary

Table 1–3 summarizes the `dropjava` arguments.

Table 1–3 dropjava Argument Summary

Argument	Description
`-user`	Specifies a user, password, and optional database connect string; the files will be dropped from this database instance.
`<filenames>`	You can specify any number and combination of `.java`, `.class`, `.sqlj`, `.ser`, `.jar`, `.zip`, and resource file names in any order.
`-oci8`	Directs `dropjava` to connect with the database using the oci8 JDBC driver. `-oci8` and `-thin` are mutually exclusive; if neither is specified, then `-oci8` is used by default. Choosing `-oci8` implies the form of the `-user` value.
`-schema`	Designates the schema from which schema objects are dropped. If not specified, the logon schema is used. To drop a schema object from a schema that is not your own, you need the DROP ANY PROCEDURE and UPDATE ANY TABLE privileges.
`-stdout`	Causes the output to be directed to `stdout`, rather than to `stderr`.
`-synonym`	Drops a PUBLIC synonym that was created with loadjava.
`-thin`	Directs `dropjava` to communicate with the database using the thin JDBC driver. `-oci8` and `-thin` are mutually exclusive; if neither is specified, then `-oci8` is used by default. Choosing -thin implies the form of the `-user` value.
`-verbose`	Directs `dropjava` to emit detailed status messages while running.

Argument Details

File Names

`dropjava` interprets most file names as `loadjava` does:

- `.class` files: `dropjava` finds the class name in the file and drops the corresponding schema object.
- `.java` and `.sqlj` files: `dropjava` finds the first class name in the file and drops the corresponding schema object.
- `.jar` and `.zip` files: `dropjava` processes the archived file names as if they had been entered on the command line.

If a file name has another extension or no extension, then `dropjava` interprets the file name as a schema object name and drops all source, class, and resource objects that match the name. For example, the hypothetical file name `alpha` drops whichever of the following exists: the source schema object named `alpha`, the class schema object named `alpha`, and the resource schema object named `alpha`. If the file name begins with the "`/`" character, then `dropjava` prepends `ROOT` to the schema object name.

If `dropjava` encounters a file name that does not match a schema object, it displays a message and processes the remaining file names.

user

`{-user | -u} <user>/<password>[@<database>]`

The permissible forms of `@<database>` depend on whether you specify `-oci8` or `-thin`; `-oci8` is the default.

- `-oci8`: `@<database>` is optional; if you do not specify, then `dropjava` uses the user's default database. If specified, then `<database>` can be a TNS name or a Net8 name-value list.
- `-thin`: `@<database>` is required. The format is `<host>:<lport>:<SID>`.
  - `<host>` is the name of the machine running the database.
  - `<lport>` is the listener port that has been configured to listen for Net8 connections; in a default installation, it is 5521.
  - `<SID>` is the database instance identifier; in a default installation, it is `ORCL`.

Here are some `dropjava` examples.

- Drop all schema objects in schema `TEST` in the default database that were loaded from `ServerObjects.jar`:

  ```
  dropjava -u SCOTT/TIGER -schema TEST ServerObjects.jar
  ```

- Connect with the thin driver, then drop a class and a resource file from the user's schema:

  ```
  dropjava -thin -u SCOTT/TIGER@dbhost:5521:orcl alpha.class beta.props
  ```

Dropping Resources

Care must be taken if you are removing a resource that was loaded directly into the server. This includes profiles if you translated on the client without using the `-ser2class` option. When dropping source or class schema objects, or resource schema objects that were generated by the server-side SQLJ translator, the schema objects will be found according to the package specification in the applicable `.sqlj` source file. However, the fully qualified schema object name of a resource that was generated on the client and loaded directly into the server depends on path information in the `.jar` file or on the command line at the time you loaded it. If you use a `.jar` file to load resources and use the same `.jar` file to remove resources, there will be no problem. If, however, you use the command line to load resources, then you must be careful to specify the same path information when you run `dropjava` to remove the resources.

Session Namespace Tools

Each database instance running the Oracle8*i* JServer software has a session namespace, which the Oracle8*i* ORB uses to activate CORBA and EJB objects. A *session namespace* is a hierarchical collection of objects known as PublishedObjects and PublishingContexts. PublishedObjects are the leaves of the hierarchy and PublishingContexts are the nodes, analogous to UNIX file system files and directories. Each PublishedObject is associated with a class schema object that represents a CORBA or EJB implementation. To activate a CORBA or EJB object, a client refers to a PublishedObject's name. From the PublishedObject, the Oracle8*i* ORB obtains the information necessary to find and launch the corresponding class schema object.

Creating a PublishedObject is known as *publishing* and can be done with the command-line publish tool or the interactive session shell, both of which this section describes. CORBA server developers create PublishedObjects explicitly after loading the implementation of an object with `loadjava`. EJB developers do not explicitly load or publish their implementations; the `deployejb` tool (see "deployejb" on page 1-89) implicitly does both.

A *PublishedObject* has the following *attributes*:

- Schema Object Name: the name of the Java class schema object associated with the PublishedObject.
- Schema: the name of the schema containing the corresponding class schema object.
- Helper Schema Object Name: the name of the helper class the Oracle8*i* ORB uses to automatically narrow a reference to an instance of the CORBA object or EJB.

PublishedObjects and PublishingContexts, as with their file and directory counterparts, have owners and rights (privileges). An owner can be a user name or a role name; only the owner can change the ownership or rights of a PublishedObject or PublishingContext. Table 1–4 describes session namespace rights.

Table 1–4 PublishingContext and PublishedObject Rights

Right	Meaning for PublishingContext	Meaning for PublishedObject
read	List contents and attributes (type, rights and creation time).	List object attributes (type, schema object, schema, helper, rights, and creation time).

Table 1–4 PublishingContext and PublishedObject Rights (Cont.)

Right	Meaning for PublishingContext	Meaning for PublishedObject
write	Create a PublishedObject or PublishingContext in the PublishingContext.	Republish object.
execute	Use contents to resolve a name.	Activate associated class.

Oracle8*i* creates a session namespace automatically when the Oracle8*i* ORB is configured. The PublishingContexts contained in Table 1–5 are present in all session namespaces:

Table 1–5 Initial PublishingContexts and Rights

Name	Owner	Read	Write	Execute
`/`	`SYS`	`PUBLIC`	`SYS`	`PUBLIC`
`/bin`	`SYS`	`PUBLIC`	`SYS`	`PUBLIC`
`/etc`	`SYS`	`PUBLIC`	`SYS`	`PUBLIC`
`/test`	`SYS`	`PUBLIC`	`PUBLIC`	`PUBLIC`

Because by default only `/test` is writable by PUBLIC, you will normally create PublishingContexts and PublishedObjects subordinate to `/test`.

The following tools support publishing and managing objects in the namespace:

Note: These tools only support Release 8.1.7. A backward compatible version for these tools are documented in Chapter 2, "Backwards Compatibility Tools".

- publish
- remove
- sess_sh

publish

The `publish` tool creates or replaces (republishes) a PublishedObject in a PublishingContext. It is not necessary to republish when you update a Java class schema object; republishing is required only to change a PublishedObject's attributes. To publish, you must have write permission (the write right) for the destination PublishingContext; by default only the PublishingContext `/test` is writable by PUBLIC. To republish you must additionally have the write permission for the PublishedObject.

Note: All supplied names are case sensitive. Thus, the schema, username, and password will not be uppercased.

Syntax

```
publish [options] <name> <class> [<helper>]
             -user | -u <username> -password |-p <password>
             -service |-s <serviceURL>
```

where options are:

```
  [-describe | -d]
  [-g | -grant {<user> | <role>}[,{<user> | <role>}]...]
  [-recursiveGrant | -rg | -rG {<user> | <role>}[,{<user> | <role>}]...]
  [-h | -help]
  [-idl]
  [-iiop]
  [-replaceIDL]
  [-resolver]
  [-role <role>]
  [-republish]
  [-schema <schema>]
  [-ssl]
  [-useServiceName]
  [-version | -v]
```

Note: If you use the `publish` as a command within sess_sh, you do not supply the user, password, or service command-line arguments.

Argument Summary

Table 1–6 summarizes the `publish` tool arguments.

Table 1–6 publish Tool Argument Summary

Option	Description
`<name>`	Name of the PublishedObject being created or republished; PublishingContexts are created if necessary.
`<class>`	Name of the class schema object that corresponds to <name>.
`<helper>`	Name of the Java class schema object that implements the `narrow()` method for `<class>`.
`-user`	Specifies identity with which to log into the database instance named in `-service`.
`-password`	Specifies authenticating password for the username specified with `-user`.
`-service`	URL identifying database whose session namespace is to be "opened" by `sess_sh`. The serviceURL has the form: `sess_iiop://`*`<host>:<lport>:<sid>`*. `<host>` is the computer that hosts the target database; `<lport>` is the listener port that has been configured to listen for session IIOP; `<sid>` is the database instance identifier. Example: `sess_iiop://localhost:2481:orcl` which matches the default installation on the invoker's machine.
`-describe`	Summarizes the tool's operation.
`-grant`	After creating or republishing the PublishedObject, grants read and execute rights to the sequence of `<user>` and `<role>` names. When republishing, replace the existing users / roles that have read / execute rights with the `<user>` and `<role>` names. To selectively change the rights of a PublishedObject, use the `sess_sh`'s `chmod` command. Note that to activate a CORBA object or EJB, a user must have the execute right for both the PublishedObject and the corresponding class schema object. The sequence of user and role names must be a comma-separated list, containing no internal spaces. **Note**: You must uppercase the schema name.
`-recursiveGrant`	Grants read and execute permission in the same manner as the -grant option; but in addition to the designated object, it also grants these permissions to all contexts that the object exists within. If the context already has a permission level of SYS, the grant for that context is ignored. You can specify either -grant or -recursiveGrant.

Table 1–6 publish Tool Argument Summary (Cont.)

Option	Description
`-help`	Summarizes the tool's syntax.
`-idl`	Load the IDL interface definition into the IFR.
`-iiop`	Connects to the target database with IIOP instead of the default session IIOP. Use this option when publishing to a database server that has been configured without session IIOP.
`-replaceIDL`	If an IDL interface definition currently exists within the IFR, replace it with this version. If not specified, the publish command will not replace the existing interface within the IFR. The -replaceIDL flag will replace any interface with the same name in the IFR, even if it was originally stored by another user. Thus, different users can overwrite another user's interface unknowingly.
`-republish`	Directs `publish` to replace an existing PublishedObject; without this option, the `publish` tool rejects an attempt to publish an existing name. If the PublishedObject does not exist, `publish` creates it. Republishing deletes non-owner rights; use the `-grant` option to add read/execute rights when republishing.
`-resolver`	Specifies an explicit resolver spec to store as part of the reference. The classloader uses this resolver spec for object activation. If -resolver is not specified, the default resolver spec, which includes current user's schema and PUBLIC, is used. See "resolver" on page 1-13 for details. When activating the object, the ORB first tries to locate all classes using the resolver spec published with the object. If the required classes are not found, the ORB then uses the caller's resolver spec.
`-role`	Role to assume for the publish; no default.
`-schema`	The schema containing the Java <class> schema object. If you do not specify, the `publish` tool uses the invoker's schema.
`-ssl`	Connects to the database with SSL server authentication. You must have configured the database for SSL to use this option, and you must specify an SSL listener port in `-service`.
`-useServiceName`	If you are using a service name instead of an SID in the URL, you must specify this flag. Otherwise, the tool assumes the last string in the URL is the SID.
`-version`	Shows the tool's version.

Here is a `publish` example.

Publish the CORBA server implementation `vbjBankTestbank.AccountManagerImpl` and its helper class as `/test/bankMgr` in the tool invoker's schema:

```
publish /test/bankMgr BankTestServer.AccountManagerImpl \
BankTestServer.AccountManagerHelper \
-user SCOTT -password TIGER \
-service sess_iiop://dlsun164:2481:orcl
```

IDL Restrictions The following are the restrictions when using the IDL options for publish:

1. You store the IDL interface definition within the IFR through the Oracle8*i* JServer `publish` command. The `publish` command stores the interface within a flat file, `AuroraIFR.idl`. Normally, this file is automatically written to `$ORACLE_HOME/javavm/admin`. However, if this directory is not write-enabled, you can specify another fully-qualified filename within the "`aurora.ifr.file`" system property through the modifyprops tool, as follows:

   ```
   modifyprops -user SCOTT/TIGER@dbhost:5521:orcl
               "aurora.ifr.file" "/private/ifr/myIFRfile"
   ```

2. The `-replaceIDL` flag will replace any interface with the same name in the IFR, even if it was originally stored by another user. Thus, different users can overwrite another user's interface unknowingly.
3. Once you have stored the interface in the IFR, there is no method for removing it. This file will continue to grow until you delete the entire file.

remove

The `remove` tool removes a PublishedObject or PublishingContext from a session namespace. It does not remove the Java class schema object associated with a PublishedObject; use `dropjava` to do that.

Note: This tool is more extensive than the `rm` command. It removes the PublishedObject or PublishingContext from the namespace and any related IFR interfaces. The `rm` command solely removes an entity from within the directory.

Syntax

```
remove <name> {-user | -u} <username> {-password | -p} <password>
              {-service | -s} <serviceURL>
[options]
  [-d | -describe]
  [-h | -help]
  [-iiop]
  [-r | -recurse]
  [-role role]
  [-ssl]
  [-useServiceName]
  [-version | -v]
```

Argument Summary

Table 1–7 describes the `remove` arguments.

Table 1–7 remove Argument Summary

Option	Description
`<name>`	Name of PublishingContext or PublishedObject to be removed.
`-user`	Specifies identity with which to log into the instance named in `-service`.
`-password`	Specifies authenticating password for the `<username>` you specified with `-user`.
`-service`	URL identifying database whose session namespace is to be “opened” by `sess_sh`. The serviceURL has the form: `sess_iiop://`*`<host>`*`:`*`<lport>`*`:`*`<sid>`*. `<host>` is the computer that hosts the target database; `<lport>` is the listener port that has been configured to listen for session IIOP; `<sid>` is the database instance identifier. Example: `sess_iiop://localhost:2481:orcl` which matches the default installation on the invoker’s machine.
`-describe`	Summarizes the tool’s operation.
`-help`	Summarizes the tool’s syntax.
`-iiop`	Connects to the target database with IIOP instead of the default session IIOP. Use this option when removing from a database server that has been configured without session IIOP.

Table 1–7 remove Argument Summary (Cont.)

Option	Description
`-recurse`	Recursively removes `<name>` and all subordinate PublishingContexts; required to remove a PublishingContext.
`-role`	Role to assume for the remove; no default.
`-ssl`	Connects to the database with SSL server authentication. You must have configured the database for SSL to use this option.
`-useServiceName`	If you are using a service name instead of an SID in the URL, you must specify this flag. Otherwise, the tool assumes the last string in the URL is the SID.
`-version`	Shows the tool's version.

Here are examples of `remove` tool usage.

- Remove a PublishedObject named `/test/testhello`:

```
remove /test/testhello -user SCOTT -password TIGER \
    -service sess_iiop://dlsun164:2481:orcl
```

- Remove a PublishingContext named `/test/etrader`:

```
remove -r /test/etrader -user SCOTT -password TIGER \
             -service sess_iiop://dlsun164:2481:orcl
```

sess_sh

The `sess_sh` (session shell) tool is an interactive interface to a database instance's session namespace. You specify database connection arguments when you start `sess_sh`. It then presents you with a prompt to indicate that it is ready for commands.

The `sess_sh` gives a session namespace much of the "look and feel" of a UNIX file system you access through a shell, such as the C shell. For example, the session shell command:

```
ls /alpha/beta/gamma
```

means "List the PublishedObjects and PublishingContexts in the PublishingContext known as `/alpha/beta/gamma`". (NT users note: `/alpha/beta/gamma`, not `\alpha\beta\gamma`.) Indeed, many session shell command names that operate on PublishingContexts have the same names as their UNIX shell counterparts that operate on directories. For example: `mkdir` (create a PublishingContext) and `cd` (change the working PublishingContext).

In addition to UNIX-style manipulation of PublishingContexts and PublishedObjects, the session shell can launch an executable, that is, a class with a static `main()` method. Executables must have been loaded with `loadjava`, but not published—publishing is for CORBA, EJB, and servlet objects only.

Syntax

```
sess_sh {-service | -s} <serviceURL> {-user | -u} <user>
        {-password  | -p} <password> [options]
  [@<filename>]
  [-batch]
  [-c | -command <command> <args>]
  [-credsFile <creds>]
  [-describe | -d]
  [-h | -help]
  [-iiop]
  [-proxy <host>:<port>]
  [-role <rolename>]
  [-ssl]
  [-useServiceName]
  [-version | -v]
```

Argument Summary

Table 1–8 summarizes the `sess_sh` command-line arguments.

Table 1–8 sess_sh Argument Summary

Option	Description

`-service | -s`	URL identifying database whose session namespace is to be "opened" by `sess_sh`. The serviceURL should contain one of the following:
	`sess_iiop://`*`<host>:<lport>:<sid>`*
	`jdbc:oracle:`*`<type>:<spec>`*
	`http://`*`<host>:<lport>`*
	`<host>` is the computer that hosts the target database;
	`<lport>` is the listener port configured to listen for session IIOP or HTTP
	`<sid>` is the database instance identifier.
	`<type>` can be either `oci8` or `thin`
	`<spec>` is the connect string, alias, or URL for the JDBC driver
	
	Examples:
	`sess_iiop://localhost:2481:orcl`
	`jdbc:oracle:thin:@dbhost:5521:ORCL`
	`http://localhost:2481`

`-user | -u`	Specifies user's name for connecting to the database. This name case insensitive; the name will always be uppercased.
`-password | -p`	Specifies user's password for connecting to the database. This name case insensitive; the name will always be uppercased.
`@<filename>`	Specifies a script file that contains sess_sh commands to be executed. See "Scripting sess_sh Commands in the @<filename> Option" on page 1-31 for structure of the indicated file.
`-batch`	Disables all messages printed to the screen. No help messages or prompts will be printed. Only responses to entered commands are printed.
`-command`	Executes the desired command. If you do not want to run sess_sh in interpretive mode, but only want to execute a single command, execute sess_sh with the -command option followed by a string that contains the command and the arguments. Once the command executes, sess_sh exits. The following executes the "ls -lR" command on the designated host: sess_sh -user SCOTT -password TIGER -service sess_iiop://dbserver:2481:orcl -command "ls -lR"
`-credsFile`	Supply a text file with credentials instead of a username and password for the connect. You create this file by exporting a wallet into a text version.
`-describe | -d`	Summarizes the tool's operation.

Table 1–8 sess_sh Argument Summary (Cont.)

Option	Description
`-echo`	Prints out every command before execution. This is useful when executing script files.
`-help`	Summarizes the tool's syntax.
`-iiop`	Connects to the target database with plain IIOP instead of the default session IIOP. Use this option for a database server configured without session IIOP.
`-proxy`	Specifies the proxy host and port number. This is only required if you are using a firewall proxy to communicate with hosts outside your internal network.
`-role`	Role to pass to database; there is no default.
`-ssl`	Connect to the database with SSL server authentication. You must have configured the database for SSL and specify an SSL port to use this option.
`-useServiceName`	If you are using a service name instead of an SID in the URL, you must specify this flag. Otherwise, the tool assumes the last string in the URL is the SID. Alternatively, you can specify USE_SERVICE_NAME in the JNDI properties instead of using this option. See the "JNDI Connections and Session IIOP Service" chapter in either the *Oracle8i CORBA Developer's Guide and Reference* or the *Oracle8i Enterprise JavaBeans Developer's Guide and Reference* for more information.
`-version`	Shows the version.

Example Here is a `sess_sh` example.

Open a session shell on the session namespace of the database `orcl` on listener port 2481 on host `dbserver`.

```
sess_sh -user SCOTT -password TIGER -service sess_iiop://dbserver:2481:orcl
```

The `sess_sh` commands span several different types of functionality, which are grouped as follows:

- sess_sh Options—describes the options for the `sess_sh` command-line tool
- Shell Commands—describes the commands that are used for manipulating and viewing contexts and objects in the namespace

- Namespace Commands—describes the commands that bind objects, manage namespace groups, and set object properties. This includes binding the UserTransaction and DataSource objects.
- Dynamic Listener Endpoint Configuration Commands—describes commands for adding endpoints to existing listeners
- Web Application Management Commands—describes commands used for managing web applications, such as servlets and Java Server Pages.

sess_sh Options

- sess_sh Tool Output Redirection
- Scripting sess_sh Commands in the @<filename> Option

sess_sh Tool Output Redirection You can specify that any output generated by the sess_sh tool is put into a file by appending the "&><*filename*>" at the end of the command options. The following pipes all output to the `listDir` file:

```
ls -lR &>/tmp/listDir
```

Scripting sess_sh Commands in the @<filename> Option This option designates a script file that contains one or more `sess_sh` commands. The script file specified is located on the client. The `sess_sh` tool reads in the file and then executes all commands on the designated server. Also, since the script file is executed on the server, any interaction with the operating system in the script file—such as redirecting output to a file or executing another script—will occur on the server. If you direct `sess_sh` to execute another script file, this file must exist within `$ORACLE_HOME` directory on the server.

You simply type in the `sess_sh` command followed by any options an any expected input arguments.

The script file contains any `sess_sh` command followed by options and input parameters. The input parameters can be passed in on the `sess_sh` command line. The `sess_sh` command processes all known `sess_sh` options and then passes on any other options and arguments to the script file.

To access arguments within the commands in the script file, place &1...&n to denote the arguments. If all input parameters are passed into a single command, you can supply a the string "&*" to denote that all input parameters are to be passed to this command.

The following shows the contents of the script file, `execShell`:

```
chmod +x SCOTT nancy /alpha/beta/gamma
chown SCOTT /alpha/beta/gamma
java testhello &*
```

Since only two input arguments are expected, you could have also implemented the `java` command input parameters as follows:

```
java testhello &1 &2
```

Note: You can also supply arguments to the `-command` option in the same manner. The following shows an example:

```
sess_sh ... -command "cd &1" contexts
```

After processing all other options, the `sess_sh` tool passes "contexts" in as the argument to the "cd" command.

To execute this file, do the following:

```
sess_sh -user SCOTT -password TIGER -service sess_iiop://dbserver:2481:orcl \
        @execShell alpha beta
```

The `sess_sh` processes all options that it knows about and passes along any other input parameters to be used by the commands that exist within the script file. In this example, the parameters, `alpha` and `beta`, are passed to the `java` command in the script file. Thus, the actual command executed is as follows:

```
java testhello alpha beta
```

You can add any comments in your script file with the hash symbol (#). The "#" symbol makes anything to the end of the line a comment, which is ignored by `sess_sh`. For example:

```
#this whole line is ignored by sess_sh
```

Shell Commands

The following shell commands behave similarly to their UNIX counterparts:

- alias
- cd
- chmod
- chown
- connect
- echo
- env
- exit
- help
- java
- ln
- ls

- mkdir
- mv
- pwd
- rm
- setenv
- version
- whoami

Each of these shell commands contains the following common options:

Table 1–9 sess_sh Command Common Options

Option	Description
`-describe	-d`	Summarizes the tool's operation.
`-help	-h`	Summarizes the tool's syntax.
`-version`	Shows the version.

alias

You can create an alias used within the script file. You can specify multiple aliases. The definition can include several commands separated on different lines. The entire definition is included within double quotes.

The syntax is as follows:

```
alias <name> <definition>
```

where the <*name*> is the alias and <*definition*> is any `sess_sh` command. For example, the following creates an alias of "`ll`" to be mapped to "`ls -l &*`"

```
alias ll "ls -l &*"
```

Thus, the command for "`ll bin webdomains`" is translated to "`ls -l bin webdomains`".

To echo an alias, execute alias with just the <*name*> and no <*description*>.

To delete an alias, execute alias with the empty string, as follows:

```
alias <name> ""
```

cd

The `cd` command is analogous to a UNIX shell's `cd` command; it changes the working PublishingContext.

Syntax

```
cd [options] [path]
```

Here is an example.

Change to root PublishingContext:

```
$ cd /
```

chmod

The `chmod` command is analogous to a UNIX shell's `chmod` command; it changes the users or roles that have rights for a PublishingContext or PublishedObject. See Table 1–4 for descriptions of the read, write, and execute rights. Only the object's owner can change its rights.

Note: All names are case sensitive. Thus, the schema, username, and password will not be uppercased.

Syntax

```
chmod [options] {+|-}{r|w|x} {<user> | <role>} [, {<user> | <role>} ...]
                             <objectname>
      [-R]
```

Argument Summary

Table 1–10 summarizes the `chmod` arguments.

Table 1–10 chmod Argument Summary

Option	Description
`+/-rwx`	Specifies the right (read, write, or execute) to be added (+) or removed (-) for `<user>` or `<role>`.
`<user> | <role>`	Specifies the user or role whose rights are to be increased or decreased. This value is case sensitive.
`<objectname>`	Specifies the name of the PublishingContext or PublishedObject whose rights are to be changed.
`-R`	Changes the execute rights recursively. This does not include symbolic links.

Here are some chmod examples.

- Give execute rights for `/alpha/beta/gamma` to SCOTT and NANCY. Note that the schemas are separated by a comma only.

```
$ chmod +x SCOTT,NANCY /alpha/beta/gamma
```

- Remove Scott's write rights for the same object:

  ```
  $ chmod -w SCOTT /alpha/beta/gamma
  ```

chown

The chown command is analogous to the UNIX chown command; it changes the ownership of a PublishingContext or PublishedObject. The owner of a newly created PublishingContext or PublishedObject is the user who publishes it. To change a PublishingContext's or PublishedObject's ownership you must be SYS.

Syntax

```
chown [options] {<user> | <role>} <objectname>
      [-R]
```

Argument Summary

Table 1–11 summarizes the `chown` arguments.

Table 1–11 chown Argument Summary

Option	Description
`<user> | <role>`	Specifies the user or role to be the new owner.
`<objectname>`	Specifies the name of the PublishingContext or PublishedObject whose owner is to be changed.
`-R`	Changes the ownership recursively. This does not include symbolic links.

Here is a `chown` example.

Make Scott the owner of `/alpha/beta/gamma`:

```
$ chown SCOTT /alpha/beta/gamma
```

connect

The `connect` tool will connect you to another server without exiting `sess_sh`. It requires the same connection options used in sess_sh. The options for `connect` are as follows:

```
connect [-options] {-user | -u} <user> {-password  | -p} <password>
                   {-service | -s} <serviceURL>
```

```
  [-credsFile <creds>]
  [-iiop]
  [-proxy <host>:<port>]
  [-role <rolename>]
  [-ssl]
  [-useServiceName]
```

Argument Summary

Table 1–8 summarizes the `connect` command-line arguments.

Table 1–12 connect Argument Summary

Option	Description

`-service | -s`	URL identifying database whose session namespace is to be "opened" by `sess_sh`. The serviceURL should contain one of the following:
	`sess_iiop://<host>:<lport>:<sid>`
	`jdbc:oracle:<type>:<spec>`
	`http://<host>:<lport>`
	`<host>` is the computer that hosts the target database;
	`<lport>` is the listener port configured to listen for session IIOP or HTTP
	`<sid>` is the database instance identifier.
	`<type>` can be either `oci8` or `thin`
	`<spec>` is the connect string, alias, or URL for the JDBC driver
	
	Examples:
	`sess_iiop://localhost:2481:orcl`
	`jdbc:oracle:thin:@dbhost:5521:ORCL`
	`http://localhost:2481`

`-user | -u`	Specifies user's name for connecting to the database. This name case insensitive; the name will always be uppercased.
`-password | -p`	Specifies user's password for connecting to the database. This name case insensitive; the name will always be uppercased.
`-credsFile`	Supply a text file with credentials instead of a username and password for the connect. You create this file by exporting a wallet into a text version.
`-iiop`	Connects to the target database with plain IIOP instead of the default session IIOP. Use this option for a database server configured without session IIOP.

Table 1–12 connect Argument Summary (Cont.)

Option	Description
`-proxy`	Specifies the proxy host and port number. This is only required if you are using a firewall proxy to communicate with hosts outside your internal network.
`-role`	Role to pass to database; there is no default.
`-ssl`	Connect to the database with SSL server authentication. You must have configured the database for SSL and specify an SSL port to use this option.
`-useServiceName`	If you are using a service name instead of an SID in the URL, you must specify this flag. Otherwise, the tool assumes the last string in the URL is the SID. Alternatively, you can specify USE_SERVICE_NAME in the JNDI properties instead of using this option. See the "JNDI Connections and Session IIOP Service" chapter in either the *Oracle8i CORBA Developer's Guide and Reference* or the *Oracle8i Enterprise JavaBeans Developer's Guide and Reference* for more information.

Examples for using connect are as follows:

Connect to an IIOP session:

```
connect -u scott/tiger -s sess_iiop://mysun:5522:ORCL
```

Connect to an HTTP SSL session through a firewall:

```
connect -u scott/tiger -s https://mysun:9090 -proxy companyx-proxy:2443
```

echo

Prints to stdout exactly what is indicated. This is used mostly in script files.

The syntax is as follows:

```
echo [<echo_string>] [<args>]
```

where *`<echo_string>`* is a string that contains the text you want written to the screen during the shell script invocation and <*`args`*> are input arguments from the user. For example, the following prints out a notification:

```
echo "Adding an owner to the schema" &1
```

If the input argument is "SCOTT", the output would be "Adding an owner to the schema SCOTT"

env

You can view environment variables and their values with the `env` command.

Syntax

```
env [<variable>]
```

Argument Summary

Table 1–13 describes the `env` arguments.

Table 1–13 env Argument Summary

Option	Description
`<variable>`	The name of the environment variable. If not provided, all variables are printed.

The following example prints out the value of the TEST variable:

```
$ env test
TEST=HELLO
```

exit

The `exit` command terminates `sess_sh`.

Syntax

```
exit
```

Here is an example:

Leave the session shell:

```
$ exit
%
```

help

The `help` command summarizes the syntax of the session shell commands. You can also use the `help` command to summarize the options for a particular command.

Syntax

```
help [<command>]
```

java

The `java` command is analogous to the JDK `java` command; it invokes a class's static `main()` method. The class must have been loaded with `loadjava`. (There is no point to publishing a class that will be invoked with the `java` command.) The `java` command provides a convenient way to test Java code that runs in the database. In particular, the command catches exceptions and redirects the class's standard output and standard error to the session shell, which displays them as with any other command output. (The usual destination of standard out and standard error for Java classes executed in the database is one or more database server process trace files, which are inconvenient and may require DBA privileges to read.)

Syntax

```
java [-schema <schema>] <class> [arg1 ... argn]
```

Argument Summary

Table 1–14 summarizes the `java` arguments.

Table 1–14 java Argument Summary

Option	Description
`class`	Names the Java class schema object that is to be executed.
`-schema`	Names the schema containing the class to be executed; the default is the invoker's schema. The schema name is case sensitive.
`arg1 ... argn`	Arguments to the class's `main()` method.

Here is a `java` command example.

Say hello and display arguments:

```
package hello;
public class World {
    public World() {
        super();
    }
    public static void main(String[] argv) {
        System.out.println("Hello from the JServer/ORB");
        if (argv.length != 0)
            System.out.println("You supplied " + argv.length + " arguments: ");
            for (int i = 0; i < argv.length; i++)
```

```
                System.out.println(" arg[" + i + "] : " + argv[i]);
    }
}
```

Compile, load, publish, and run the executable as follows, substituting your userid, host, and port information as appropriate:

```
% javac hello/World.java
% loadjava -r -user SCOTT/TIGER@localhost:2481:orcl hello/World.class
% sess_sh -user SCOTT -password TIGER -service sess_iiop://localhost:2481:orcl
$ java testhello alpha beta
Hello from the JServer/ORB
You supplied 2 arguments:
arg[0] : alpha
arg[1] : beta
```

ln

The `ln` (link) command is analogous to the UNIX `ln` command. A link is a synonym for a PublishingContext or PublishedObject. A link can prevent a reference to a PublishingContext or PublishedObject from becoming invalid when you move a PublishingContext or PublishedObject (see "mv" on page 1-43); creating a link with the old name makes the object accessible by both its old and new names.

Syntax

```
ln [-symbolic | -s] <object> <link>
```

Argument Summary

Table 1–15 summarizes the `ln` arguments.

Table 1–15 ln Argument Summary

Option	Description
`-s`	Create a symbolic soft link for the <object> of the <link> name.
`<object>`	The name of the PublishingContext or PublishedObject for which a link is to be created.
`<link>`	The synonym by which `<object>` is also to be known.

Here is an `ln` command example.

Preserve access via `old` although the object's name is changed to `new`:

```
$ mv old new
$ ln new old
```

ls

The `ls` (list) command shows the contents of PublishingContexts as the UNIX `ls` command shows the contents of directories.

Syntax

```
ls [options] [<pubcon> | <pubobj [<pubcon> | <pubobj] ...]
  [-dir]
  [-l]
  [-ld | ldir]
  [-R]
```

Argument Summary

Table 1–16 describes the `ls` arguments.

Table 1–16 ls Argument Summary

Option	Description
`<pubcon> | <pubobj>`	Name of PublishingContext(s) and / or PublishingObject(s) to be listed; the default is the working PublishingContext.
`-dir`	Shows only PublishingContexts; analogous to the UNIX `ls -d` command.
`-l`	Shows contents in long (detailed) format. The long format includes name, creation time, owner, and rights. For PublishedObjects, the option also shows class, schema, and helper. You can use this option in conjunction with -R, as -lR or -Rl.
`-ldir`	Lists PublishingContexts in long format, ignoring PublishingObjects; analogous to UNIX `ls -ld` command.
`-R`	Lists recursively. You can use this option in conjunction with -l, as -lR or -Rl.

Here are examples of the `ls` command.

Show contents of the root PublishingContext in short format:

```
$ ls /
bin/
etc/
```

```
test/
```

Show contents of the root PublishingContext in long format:

```
$ ls -l /
Read    Write   Exec      Owner   Date   Time    Name      Schema    Class     Helper
PUBLIC  SYS     PUBLIC    SYS     Dec 14 14:59  bin/
PUBLIC  SYS     PUBLIC    SYS     Dec 14 14:59  etc/
PUBLIC  PUBLIC  PUBLIC    SYS     Dec 14 14:59  test/
```

Show contents of the /test PublishingContext in long format:

```
$ ls -l test
Read  Write Exec  Owner Date   Time  Name Schema Class                  Helper
SCOTT SCOTT SCOTT SCOTT Dec 14 16:32 bank SCOTT  Bank.AccountManagerImpl Bank.AccountManagerHelper
```

mkdir

The `mkdir` command is analogous to the UNIX shell `mkdir` command; it creates a PublishingContext. You must have the write right for the target PublishingContext to use `mkdir` in it.

Syntax

```
mkdir [options] <name>
 [-path | -p]
```

Argument Summary

Table 1–17 describes the `mkdir` arguments.

Table 1–17 mkdir Argument Summary

Option	Description
`<name>`	Name of PublishingContext to create.
`-path`	Creates intermediate PublishingContexts if they do not exist.

Here are examples of the `mkdir` command.

Create a PublishingContext called `/test/alpha` (`/test` exists):

```
mkdir /test/alpha
```

Create a PublishingContext called `/test/alpha/beta/gamma` (`/test/alpha/beta` does not exist):

```
$ mkdir -path /test/alpha/beta/gamma
```

mv

The mv command is analogous to the UNIX shell mv command.

Syntax

```
mv [options] <old> <new>
```

Here is an example of the mv command.

Change the name of /test/foo to /test/bar:

```
$ mv /test/foo /test/bar
```

pwd

The pwd command displays the name of the current working PublishingContext. It is analogous to the UNIX pwd command.

Syntax

```
pwd [options]
```

Here is an example of the pwd command.

```
$ pwd
/test/alpha
```

rm

The rm command is analogous to the rm -r UNIX shell command; it removes a PublishedObject or a PublishingContext, including its contents. To remove an object, you must have the write right for the containing PublishingContext.

Syntax

```
rm [options] <object> ... <object>
   [-r]
```

Argument Summary

Table 1–18 describes the rm arguments.

Table 1–18 rm Argument Summary

Option	Description
`<object>`	Name of PublishedObject or PublishingContext to be removed.

Table 1–18 rm Argument Summary (Cont.)

Option	Description
`-r | -recurse`	Interprets `<object>` as a PublishingContext; removes it and its contents recursively.

Here is an example of the `rm` command.

Remove the PublishedObject `/test/bank`:

```
rm /test/bank
```

Remove the PublishingContext `/test/release3` and everything it contains:

```
rm -r /test/release3
```

setenv

You can set environment variables within a script or for use within the current invocation of the sess_sh tool. These variables are not valid outside of sess_sh and are lost when sess_sh terminates.

Syntax

```
setenv <variable> <value>
```

Argument Summary

Table 1–19 describes the `setenv` arguments.

Table 1–19 setenv Argument Summary

Option	Description
`<variable>`	The name of the environment variable.
`<value>`	The value to set the environment variable to. If no value is given, the defined <variable> is removed.

The following example sets the TEST variable to the string HELLO. Once set, the value is shown with the `env` command.

```
$ setenv TEST HELLO
$ setenv PATH .:/bin:/test/bin
$ env test
TEST=HELLO
PATH=.:/bin:/test/bin
```

To remove an environment variable, set the variable to the NULL string. The following removes the TEST variable:

```
$ setenv TEST ""
```

version

The `version` command shows the version of the `sess_sh` tool. You can also show the version of a specified command.

Syntax

```
version [options] [<command>]
```

Here is an example of the `version` command.

Display the session shell's version:

```
$ version
1.0
```

whoami

Prints out the current user that logged into this session.

Namespace Commands

- addgroupentry
- bind
- bindut
- bindds
- getgroup
- getproperties
- publish
- removegroupentry
- setgroup
- setproperties

addgroupentry

The `addgroupentry` command adds a single property to an existing property group for the designated JNDI object.

Syntax

```
addgroupentry <object_name> <group_name> <prop_name> <prop_value>
```

Table 1–20 addgroupentry Command Options

Option	Description
`<object_name>`	A JNDI name that is bound to an object.
`<group_name>`	A property group name, which was created by the setgroup command.
`<prop_name>`	The property name assigned to the group / object.
`<prop_value>`	The value for the property

The following example sets another property for the `wine` group of the `config` object:

```
addgroupentry config wine type merlot
```

bind

The `bind` command binds an object reference or a naming context into the JNDI namespace. The ordering of the options must be in the order specified here. You cannot mix the options.

Syntax

```
bind <JNDI_object_name> [options]
   [-context]
   [-rebind]
   {-class | -c <classname>
   [-factory | -f <factory>]
   [-location | -l <URL>]
   [-string <type_name> <string_value> [-string <type_name> <string_value> ...]]
   [-binary <type_name> <string_value> [-binary <type_name> <string_value> ...]]
```

Table 1–21 bind Command Options

Option	Description
`<JNDI_object_name>`	The JNDI name that the object is bound to within the namespace. This is the name that is used to retrieve the bound object.
`-context`	The object to be bound is a JNDI Context or InitialContext.
`-rebind`	If the JNDI name already exists, replace the object that it is bound to with this object.
`-class`	Specify the class name of the bound object.

Table 1–21 bind Command Options (Cont.)

Option	Description
`-factory`	Specify the factory name for creating the object. JNDI uses this for creating the object.
`-location`	Specify the factory location if the default location is not used. This takes a JNDI URL. Refer to the JNDI specification for more information.
`-string`	Specify a String reference attribute for the object by the type name and value.
`-binary`	Specify a Binary reference attribute for the object by the type and a binary value. The given string value is converted into binary.

The following binds a CORBA IOR reference into the JNDI namespace. The object reference was stringified before the `bind` is executed and is substituted for the input argument $1. In addition, a binary reference attribute for the employee site number of 400 is also bound within the object.

```
bind /test/employee -class employee.Employee -factory employee.EmployeeFactory
          -string EmpObjRef $1 -binary EmpNumber 400
```

bindut

The `bindut` command binds a `UserTransaction` object in the namespace. You must bind a `UserTransaction` object for both single and two-phase commit transactions. In a two-phase commit scenario, the `UserTransaction` is bound with respect to the two-phase commit engine.

> **Note:** If you change the two-phase commit engine, you must update all database links on all `DataSource` objects involved in the transaction, and rebind the `UserTransaction`.

Syntax

```
bindut <lookup_name> [options]
      [-help | -h]
      [-describe | -d]
      [-version | -v]
      [-rebind]
      [-expprop]
      [-host <hostname> -port <portnum> -sid <SID>]
      [-url <db_url>]
```

```
[-g | -grant {<user> | <role>}[,{<user> | <role>}]...]
[-recursiveGrant | -rg | -rG {<user> | <role>}[,{<user> | <role>}]...]
[-user | -u <user>]
[-password | -p <password>]
```

Argument Summary

Table 1–22 summarizes the `bindut` command-line arguments:

Table 1–22 bindut Argument Summary

Option	Description
`<lookup_name>`	The JNDI name of the UserTransaction object
`-help`	Summarizes the tool's syntax.
`-describe`	Summarizes the tool's operation.
`-version`	Shows the tool's version.
`-rebind`	If the JNDI name for the UserTransaction object already exists, you must specify this option if you want it overwritten with this new object. Otherwise, no bind will occur for this option and an AlreadyBound exception is thrown.
`-expprop`	Specify this option only if host/port/sid options are specified. Designates how the transaction is propagated between objects. If an IIOP client invokes an IIOP server method, the transaction context is propagated implicitly for you. However, if your client uses JDBC or HTTP for communicating, the propagation context must be propagated explicitly. Specify this flag in the case of JDBC or HTTP communication.

`-host <host>`
`-port <port>`
`-sid <sid>`
	These options specify the Oracle8*i* database that is acting as the two-phase commit engine. These are only necessary for any global transactions that use two-phase commit. You can either specify the two-phase commit engine location either through these options or within the -url option. The default value for -sid is ORCL.

`-url <db_url>`	This URL specifies the location of the Oracle8*i* database that is acting as the two-phase commit engine. You can specify the two-phase commit engine either through this option or by specifying each part of the URL separately within the four options mentioned above. This URL can be either JDBC Thin or sess_iiop URL.

Table 1–22 bindut Argument Summary (Cont.)

Option	Description
`-grant <user> | <role>`	Grants read and execute rights to the sequence of <user> and <role> names. When rebinding any leaf nodes, replace the existing users/roles that have read/execute rights with the <user> and <role> names. To selectively change the rights of a UserTransaction object, use the sess_sh's chmod command. The sequence of user and role names must be a comma-separated list, containing no internal spaces. **Note**: You must uppercase the schema name.
`-recursiveGrant <user> | <role>`	Grants read and execute permission like the -grant option to the designated object and to all contexts that the object exists within. If the context has a permission level of SYS, the grant for that context is ignored. You can specify either -grant or -recursiveGrant.
`-user | -u <user>`	Specifies user's name for connecting to the two-phase commit engine. This option is only required for two-phase commit scenario.
`-password | -p <password>`	Specifies user's password for connecting to the two-phase commit engine. This option is only required for two-phase commit scenario.

Example The following example binds the `ut1 UserTransaction` within the namespace designating the two-phase commit engine at `dbsun.mycompany.com`:

```
bindut /test/UserTransaction/ut1 -host dbsun.mycompany.com -port 2481 -sid ORCL
```

The same command could be issued as follows:

```
bindut /test/UserTransaction/ut1 -url sess_iiop://dbsun.mycompany.com:2481:ORCL
```

The options used to bind a `UserTransaction` object depend on whether the transaction uses single or two-phase commit, as described below:

- Single-phase commit—provide the JNDI bound name for the `UserTransaction` object. You do not need to provide the address to a two-phase commit engine. For example, the following binds a `UserTransaction` with the name of "`/test/myUT`" that exists for a single-phase commit transaction:

  ```
  bindut /test/myUT
  ```

- Two-phase commit—provide the JNDI bound name for the `UserTransaction` object and the address to a two-phase commit engine. For example, the following binds a `UserTransaction` with the name of "`/test/myUT`" and a two-phase commit engine at "`2pcHost`":

  ```
  bindut /test/myUT -url jdbc:oracle:thin:@2pcHost:5521:ORCL
  ```

  When the transaction commits, the `UserTransaction` communicates with the two-phase engine designated in the `-url` option to commit all changes to all included databases. The `UserTransaction` tracks all databases involved in the transaction; the two-phase commit engine uses the database links for these databases to complete the transaction.

bindds

The `bindds` command binds a `DataSource` object in the JNDI namespace. In order to enlist any database—including the local database—you must bind a JTA `DataSource` object to identify each database included in the transaction. If you require a two-phase commit transaction, your system administrator must create public database links from the two-phase commit engine to each database involved in the transaction. These database link names are included when binding `DataSource` objects.

Note: If you change the two-phase commit engine, you must update all database links on all `DataSource` objects involved in the transaction, and rebind the `UserTransaction`.

For JTA, XA, or JNDI, you might need to bind a `DataSource` object in the JNDI namespace for later retrieval and activation of a database connection. There are four types of `DataSource` objects that you can bind using the `bindds` command.

- `OracleDataSource`—a `DataSource` object modified for use with an Oracle8*i* database.
- `OracleJTADataSource`—a `DataSource` object modified for use within global JTA transactions. Within JTA, in order to enlist any database—including the local database—you must bind an `OracleJTADataSource` object to identify each database included in the transaction.
- `OracleConnectionPoolDataSource`—a `DataSource` object modified for use with a pool of `DataSource` objects.

- `OracleXADataSource`—a `DataSource` object modified for use within an XA-type connection.

The `DataSource` object type is specified with the `-dstype` option of the `bindds` command, as described below:

Syntax

```
bindds <lookup_name> [options]
       [-help | -h]
       [-describe | -d]
       [-version | -v]
       [-dstype <datasource>]
       [-host <hostname> -port <portnum> -sid <SID> -driver <driver_type>]
       [-url <db_url>]
       [-dblink <DBLINK>]
       [-g | -grant {<user> | <role>}[,{<user> | <role>}]...]
       [-recursiveGrant | -rg {<user> | <role>}[,{<user> | <role>}]...]
       [-rebind]
       [-user | -u <user>]
       [-password | -p <password>]
```

Argument Summary

Table 1–23 summarizes the `bindds` command-line arguments:

Table 1–23 bindds Argument Summary

Option	Description
`<lookup_name>`	The JNDI name of the DataSource object
`-help`	Summarizes the tool’s syntax.
`-describe`	Summarizes the tool’s operation.
`-version`	Shows the tool’s version.
`-dstype <datasource>`	The type of DataSource object that you are currently binding. Values can be one of the following: ■ do not specify this option to bind an OracleDataSource ■ jta—OracleJTADataSource ■ xa—OracleXADataSource ■ pool—OracleConnectionPoolDataSource If you do not specify this option, the default is an OracleDataSource object.

Table 1–23 bindds Argument Summary (Cont.)

Option	Description

`-host <host>`
`-port <port>`
`-sid <sid>`
`-driver <drv_type>`
	These options specify the location of the database and driver type for the connection to be established to the database. This information enables anyone retrieving this object to establish a connection to this database. You can alternatively specify this information within a URL format within the -url option. The default value for -sid is ORCL. Values for -driver can be thin, oci8, or kprb.

`-url <db_url>`	This JDBC URL specifies the location of the database. With this information bound within the DataSource object, a connection can be created to this database. Alternatively, you can specify this information within the four options mentioned above. You must specify a JDBC URL; an IIOP (sess_iiop) address is not permitted.
`-dblink <DBLINK>`	The fully-qualified database link, which must be previously configured by an administrator, from the two-phase commit engine to the database described by this DataSource object. This option is only necessary for two-phase commit transactions. The public database link must be previously created by a system administrator on the two-phase commit engine.

`-grant`
`<user> | <role>`
	Grants read and execute rights to the sequence of <user> and <role> names. When rebinding, replace the existing users/roles that have read/execute rights with the <user> and <role> names. To selectively change the rights of a DataSource, use the sess_sh's chmod command. The sequence of user and role names must be a comma-separated list, containing no internal spaces.
	Note: You must uppercase the schema name.

`-recursiveGrant`
`<user> | <role>`
	Grants read and execute permission like the -grant option to the designated object and to all contexts that the object exists within. If the context has a permission level of SYS, the grant for that context is ignored. You can specify either -grant or -recursiveGrant.

`-rebind`	If the DataSource object already exists, you must specify this option if you want it overwritten with this new object. Otherwise, no bind will occur for this option.
`-user | -u <user>`	Specifies user's name for connecting to the database. Stores the username within the DataSource object. If no username is supplied within the JNDI Context when creating the database connection, this username is used.

Table 1–23 bindds Argument Summary (Cont.)

Option	Description
`-password | -p <password>`	Specifies user’s password for connecting to the database. Stores the password within the DataSource object. If no password is supplied within the JNDI Context when creating the database connection, this username is used.

Binding an OracleJTADataSource Object You bind an `OracleJTADataSource` object for any databases included in a global transaction. If you require a two-phase commit transaction, your system administrator must create public database links from the two-phase commit engine to each database involved in the transaction. These database link names must be included when binding the `OracleJTADataSource` object.

Note: In a two-phase commit scenario, the `DataSource` object is bound with respect to the two-phase commit engine. If you change the two-phase commit engine, you must update all database links, and rebind all concerned `DataSource` objects.

The following example binds the `ds1` `OracleJTADataSource` into the namespace with `ds1db` as the database link name created on the two-phase commit engine:

```
% bindds /test/ds1 -host dbsun.mycompany.com -port 2481
                   -sid ORCL -driver thin -dstype jta -dblink ds1db.oracle.com
```

The options used to bind an `OracleJTADataSource` object depend on whether the transaction uses single or two-phase commit, as described below:

- Single-phase commit—provide the JNDI bound name and the URL address information for this database within the `OracleJTADataSource` object. You do not need to provide a database link. For example, the following binds an `OracleJTADataSource` with the name of "`/test/myUT`" that exists within a single-phase commit transaction:

  ```
  bindds /test/ds1 -host dbsun -port 5521 -sid ORCL -driver thin -dstype jta
  ```

- Two-phase commit—provide the JNDI bound name for the object, the URL for creating a connection to the database, and the database link from the two-phase commit engine to this database.

  ```
  bindds /test/myUT -url jdbc:oracle:thin:@dbsun:5521:ORCL -dstype jta
        -dblink mydsdblink.oracle.com
  ```

This includes not only the information for creating a connection to this database, but also the information needed by the two-phase commit engine to facilitate committing a global transaction.

getgroup

The `getgroup` command lists all of the properties within a property group for the designated JNDI object.

Syntax

```
getgroup <object_name> <group_name>
```

Table 1–24 getgroup Command Options

Option	Description
`<object_name>`	A JNDI name that is bound to an object.
`<group_name>`	A property group name, which was created by the setgroup command.

The following example displays all properties defined for the `wine` group of the `config` object:

```
getgroup config wine
```

getproperties

The `getproperties` command lists all properties associated with the given JNDI name.

Syntax

```
getproperties [-all] <object_name>
```

Table 1–25 getproperties Command Options

Option	Description
`-all`	Display all properties information including the reference information, such as the class, factory, and factory location.
`<object_name>`	A JNDI name that is bound to an object.

publish

The `publish` command performs the same function as the publish tool. Refer to "publish" on page 1-21 for command syntax and examples.

removegroupentry

The `removegroupentry` command removes a single property to an existing property group for the designated JNDI object.

Syntax

```
removegroupentry <object_name> <group_name> <prop_name>
```

Table 1–26 removegroupentry Command Options

Option	Description
`<object_name>`	A JNDI name that is bound to an object.
`<group_name>`	A property group name, which was created by the setgroup command.
`<prop_name>`	The property name assigned to the group / object.

The following example removes the `type` property from the `wine` group of the `config` object:

```
removegroupentry config wine type
```

setgroup

The `setgroup` command creates a property group for a JNDI object. You add properties to an existing group through the addgroupentry command. Each execution of `setgroup` either creates a new group or overwrites an existing group. To specify multiple properties, enclose all name-value pairs within double-quotes (") and separate each name-value pair with a newline.

Syntax

```
setgroup <object_name> <group_name> "<prop_name=prop_value>
          [<prop_name=prop_value>...]"
```

Table 1–27 setgroup Command Options

Option	Description
`<object_name>`	A JNDI name that is bound to an object.

Table 1–27 setgroup Command Options (Cont.)

Option	Description
`<group_name>`	The property group name to be used for categorizing the given properties.
`<prop_name>`	The property name that has already been created with the setproperties command.
`<prop_value>`	The value for the property

The following example sets three properties for `wine` group in the `config` object:

```
setgroup config wine "debug=true
>servlet.class=SCOTT:winemasters.tasting.Tasting
>details=high"
```

setproperties

The `setproperties` command assigns name-value pairs to an object with the given JNDI name. Each execution of `setproperties` resets all properties for this object to what is indicated on the command-line. To specify multiple properties, enclose all name-value pairs within double-quotes (") and separate each name-value pair with a newline.

Syntax

```
setproperties <object_name> "<prop_name=prop_value>
          [<prop_name=prop_value>...]"
```

Table 1–28 setproperties Command Options

Option	Description
`<object_name>`	A JNDI name that is bound to an object.
`<prop_name>`	The name of the property.
`<prop_value>`	The current value for the property.

The following example sets three properties for the `config` object:

```
setproperties config "debug=true
>servlet.class=SCOTT:winemasters.tasting.Tasting
>details=high"
```

Dynamic Listener Endpoint Configuration Commands

The following sess_sh commands are provided to modify an existing listener.

- regep
- unregep

regep

In order to receive incoming requests, the listener must be configured with an endpoint for each presentation type. Most listeners are configured to accept Net8 (`TTC`) connections. The other two types of supported presentations are IIOP (`oracle.aurora.server.SGiopServer`) and HTTP (`HTTP://webserver`). In addition, if you create your own presentation, the listener must have an endpoint registered for that presentation.

You can either statically configure these endpoints within the listener configuration (either through the Net8 configuration tool or by modifying the listener configuration file) or dynamically register these endpoints through the `regep` tool.

The register endpoint (`regep`) command dynamically registers an endpoint within the existing listener for the specified presentation type. For example, you can modify a listener that exists primarily for TTC requests to also accept IIOP requests.

This tool requires you log on as a system user.

Syntax

```
regep -pres <presentation_string> [-host <hostname>] -port <portnum>
      [-listener <lsnr_addr>]
      [-ssl]
      [-persistent]
      [-describe | -d]
      [-help | -h]
      [-version | -v]
```

Argument Summary

Table 1–29 summarizes the `regep` command-line arguments:

Table 1–29 regep Argument Summary

Option	Description
`-pres`	A presentation string. For IIOP requests, this string is "`oracle.aurora.server.SGiopServer`". For HTTP requests, this string is "`HTTP://webserver`"
`-host`	Specifies the hostname or IP address where the endpoint is to be registered. If you omit this option or supply "*" as the value, this endpoint will listen on all IP interfaces for the host. That is, you can connect using the IP address, the host name, or the localhost logical name. If you choose to listen on all IP interfaces, lsnrctl will return only one of these values.
`-port`	Specifies the port number for the endpoint. Must be a valid port number.
`-listener`	If specified, defines a listener with the specified address. The address equals the string given within the "address=" portion of the listener configuration string. If unspecified, the local listener is used.
`-ssl`	If specified, the endpoint is defined as an SSL endpoint.
`-persistent`	If specified, the endpoint is made persistent. The default is non-persistent. A persistent endpoint re-registers itself after the database has been restarted.
`-describe | -d`	Summarizes the tool's operation.
`-help | -h`	Summarizes the tool's syntax.
`-version | -v`	Shows the version.

unregep

The unregister endpoint (`unregep`) command unregisters an existing dynamic listener endpoint. This tool requires you log on as a system user.

Syntax

```
unregep -pres <presentation_string> -host <hostname> -port <portnum>
      [-describe | -d]
      [-help | -h]
      [-version | -v]
      [-delete]
```

Argument Summary

Table 1–30 summarizes the `unregep` command-line arguments:

Table 1–30 unregep Argument Summary

Option	Description
`-pres`	A presentation string. For IIOP requests, this string is "`oracle.aurora.server.SGiopServer`". For HTTP requests, this string is "`HTTP://webserver`"
`-host`	Specifies the hostname where the endpoint is to be unregistered.
`-port`	Specifies the port number for the endpoint.
`-describe | -d`	Summarizes the tool's operation.
`-help | -h`	Summarizes the tool's syntax.
`-version | -v`	Shows the version.
`-delete`	Deletes the endpoint completely. If not specified, this endpoint will be registered upon database startup.

Web Application Management Commands

The session shell provides a set of specialized commands for manipulating the OSE JNDI namespace, the web server and publish servlets. The uses and syntax requirements of each command are described in the following sections:

- Service Configuration—Create the service root for a designated service within the namespace.
- Web Domain Configuration—Create one or more domains within the service root.
- Servlet Context Management—Create servlet contexts in one of the defined domains.
- Servlet Management—After copying the servlets into the servlet context, you publish it so that the clients can access it.
- JavaServer Pages Management—You must publish any JavaServer Pages before a client can access them.

- Export Commands—Export configuration information about a web domain into a mod_OSE format.
- Security Management—Specify the security realm.

Service Configuration

The following session shell commands are provided to create a new service. Each service is associated with a different presentation string and protocol.

- addendpoint
- createservice
- createwebservice
- destroyservice
- rmendpoint

addendpoint

Adds a new endpoint dynamically with the database listener. The listener must already exist. This command only registers a new endpoint for a web service with the existing listener.

Syntax

```
addendpoint [options] <service> <name>
        [-listener <lsnr>]
        [-net8]
        [-interface <int_spec>]
        [-port <port_num>]
        [-register]
        [-ssl]
        [-threads <min> <max>]
        [-timeout <seconds>]
```

Argument Summary

Table 1–31 summarizes the `addendpoint` command-line arguments:

Table 1–31 addendpoint Command Options

Option	Description
`<service>`	The service that the endpoint will listen for incoming requests on. For example, `webservice` is a valid service name.
`<name>`	The name of the endpoint.

Table 1–31 addendpoint Command Options (Cont.)

Option	Description
`-listener <lsnr>`	The address of the listener to add the endpoint to for this service. If not specified, the endpoint is added to the default listener.
`-net8`	Specifies that the endpoint is configured to accept requests over the Net8 protocol. If not specified, the endpoint is configured to accept requests over the TCP protocol. Use Net8 when using mod_OSE to communicate with Servlets.
`-port <port_num>`	The port number that the endpoint is registered for listening on.
`-interface <int_spec>`	The IP address used to connect to this service. The default allows all IP addresses. The IP address specified is mapped to the service domain.
`-register`	This option specifies that the endpoint should always exist. That is, every time this listener is initiated, the endpoint exists on that listener. If not specified, the endpoint terminates when the listener terminates.
`-ssl`	Specifies that the endpoint listens for secure requests that use the SSL protocol.
`-threads <min> <max>`	The minimum and maximum number of threads for the endpoint. The minimum value is started upon listener initialization; the maximum value is used to deny any more incoming requests.
`-timeout <seconds>`	The socket read timeout value in seconds. The amount of time that the web server will allow you to block on the socket.

The following example adds a listener endpoint on port 8080 for the `webserver` service. It starts up three threads and has a socket read timeout of 300 seconds.

```
addendpoint -port 8080 -threads 3 5 -timeout 300 webserver endpt1
```

createservice

Creates basic information for a service that is used during service installation. The service can either be an HTTP or IIOP based service, or a brand-new service that you have developed.

Syntax

```
createservice [options] <service>
        [-http | -iiop]
        [-service <class>]
        [-properties <prop_groups>]
        [-root <location>]
        [-globalTimeout <seconds>]
```

Argument Summary

Table 1–32 summarizes the `createservice` command-line arguments:

Table 1–32 createservice Argument Summary

Option	Description
`<service>`	The user-defined name of the new service.
`-http`	The service is HTTP-based.
`-iiop`	The service is IIOP-based.
`-service <class>`	The Java class that implements the `<service>`. Defaults to Oracle-provided classes if specifying `-http` or `-iiop`. Other presentations require this option.
`-properties <prop_groups>`	List of property groups to use as the defaults for this service. Specify the name-value pairs in the same way as in the setgroup command.
`-root <location>`	JNDI location for the service configuration.
`-globalTimeout <seconds>`	Timeout for database sessions processing requests for this service. Timeout is specified in seconds.

createwebservice

Creates basic information for a web-based service that is used during service installation. This service uses the HTTP protocol for connecting. This is the similar to executing `createservice -http`.

Syntax

```
createwebservice [options] <service_name>
        -root <location>
        [-properties <prop_groups>]
        [-ip]
        [-virtual]
```

Argument Summary

Table 1–33 summarizes the `createwebservice` command-line arguments:

Table 1–33 createwebservice Argument Summary

Option	Description
`<service>`	The user-defined name of the new service.
`-root <location>`	JNDI location for the web service configuration.
`-properties <prop_groups>`	List of property groups to use as the defaults for this service. Specify the name-value pairs in the same way as in the setgroup command.
`-ip`	The web service allows IP-based multi-homed domains.
`-virtual`	The web service allows VirtualHost multi-homed domains.

Note: If neither `-ip` or `-virtual` are specified, the web service is a single domain web service. The name given will be the service root name.

The following example creates a web service, `webserver`, that is defined in the "`/webdomains`" directory. The "`/webdomains`" directory should have been created by the createwebdomain command.

```
createwebservice -root /webdomains webserver -properties "debug=true
>servlet.class=SCOTT:customer.CustMain
>details=default"
```

destroyservice

Removes a defined service (created either by the createservice or createwebservice commands), unregisters all endpoints, and removes the endpoints so that they will not be started when the listener is initiated again.

```
destroyservice [-all] <service_name>
```

Argument Summary

Table 1–34 summarizes the `destroyservice` command-line arguments:

Table 1–34 destroyservice Command Options

Option	Description
`<service>`	The service created by the createservice or createwebservice commands.
`-all`	Erases everything under this service root.

The following example deletes the `webserver` service.

```
destroyservice webserver
```

rmendpoint

Removes a specific endpoint from a service and unregisters it from the listener.

Syntax

```
rmendpoint <service> <name>
```

Argument Summary

Table 1–35 summarizes the `rmendpoint` command-line arguments:

Table 1–35 rmendpoint Command Options

Option	Description
`<service>`	The service that the endpoint will listen for incoming requests on. For example, `webservice` is a valid service name.
`<name>`	The name of the endpoint that was created with addendpoint.

The following example deletes the `endpt1` endpoint:

```
rmendpoint webserver endpt1
```

Web Domain Configuration

Within the service root, you create one or more web domains. These web domain store servlet contexts. Use web domains to organize your servlet contexts. These commands enable you to create and destroy any web domain.

- createwebdomain
- destroywebdomain

createwebdomain

Creates a web domain owned by the specified schema. This domain contains services. The services contain servlets, which execute under the schema's identity. Each web domain is initialized with the "`/default`" servlet contexts.

Syntax

```
createwebdomain [options] <domain_name>
        [-docroot <location>]
        [-properties <prop_groups>
```

Argument Summary

Table 1–36 summarizes the `createwebdomain` command-line arguments:

Table 1–36 createwebdomain Command Options

Option	Description
`<domain_name>`	The full path of where the domain should be located and its name.
`-docroot <location>`	The location of the servlet static pages for this webdomain's default context. Other context's docroot location is specified in the createcontext command.
`-properties <prop_groups>`	List of property groups to use as the defaults for this service. Specify the name-value pairs in the same way as in the setgroup command.

The following command creates the `/webserver` domain in the root directory.

```
createwebdomain /mywebserver
```

destroywebdomain

Removes the web domain created by the `createwebdomain` command. This command also deletes any servlet contexts contained within the domain.

Syntax

```
destroywebdomain <domain_name>
```

Argument Summary

Table 1–37 summarizes the `destroywebdomain` command-line arguments:

Table 1–37 destroywebdomain Command Options

Option	Description
`<domain_name>`	The full directory and name used within `createwebdomain` to create the domain.

The following example deletes the `/webserver` domain and all servlets contained within it:

```
destroywebdomain /webserver
```

Servlet Context Management

Once all domains are setup, you can create the servlet context to exist within the specified domain. Once created, you can copy servlets into each context.

Management commands for servlet contexts include the following:

- accesslog
- adderrorpage
- createcontext
- destroycontext
- realm
- rmerrorpage

accesslog

Specifies how HTTP access logging is handled for the servlet context. This records information about each incoming HTTP request. You have one of three options.

Syntax

```
accesslog [options] <context_name>
       [-trace | -systable | -table <table_spec>]
```

Argument Summary

Table 1–38 summarizes the `accesslog` command-line arguments:

Table 1–38 accesslog Command Options

Option	Description
`<context_name>`	The name of the servlet context.
`-trace`	Write all log entries to the .TRC text file.

Table 1–38 accesslog Command Options (Cont.)

Option	Description
`-systable`	Write all log entries to the SYS.JAVA$HTTP$LOG$ table. This is the default logging option. The owner of the service context must have permission to access this table. If not, specify the -table option with a table that the owner does have permission for.
`-table <table_spec>`	Write all log entries to the designated table. The table must contain the same layout as the SYS.JAVA$HTTP$LOG$ table.

The following example specifies that the HTTP access log messages for the `/webdomains/contexts/default` service context should be directed into the `HTTP_LOG` table in SCOTT's schema:

```
accesslog -table SCOTT:HTTP_LOG /webdomains/contexts/default
```

adderrorpage

When a specific error code is returned, you can specify a URL that the client is directed to. This is useful for displaying error messages to the client.

Syntax

```
adderrorpage -error <errcode> -virtualpath <errorpath> <context_name>
```

Argument Summary

Table 1–39 summarizes the `adderrorpage` command-line arguments:

Table 1–39 adderrorpage Command Arguments

Arguments	Description
`<context_name>`	The directory path and name of the servlet context.
`-error <errcode>`	The error code that identifies the error page.
`-virtualpath <errorpath>`	The error page, which is a servlet virtual path that this error code is associated with. This is a URI that can map to a servlet, which will be served up, or can map to a static HTML page. The web server serves up whatever the URI maps to.

The following example associates the error 401 with the servlet identified within the `-virtualpath` option. This code is valid for the default context.

```
adderrorpage -error 401 -virtualpath /SCOTT/Errors/Err401 \
             /webdomains/context/default
```

createcontext

Creates a servlet context within the specified domain, which was created by the createwebdomain command.

Syntax

```
createcontext [options] <domain_name> <context_name>
    -virtualpath <path>
    [-recreate]
    [-properties <prop_groups>]
    [-docroot <location>]
    [-stateless]
```

Argument Summary

Table 1–40 summarizes the `createcontext` command-line arguments:

Table 1–40 createcontext Command Options

Option	Description
`<domain_name>`	The directory and name for the domain where the servlet context is to be created. This domain must already exist. You create the domain through the createwebdomain command.
`<context_name>`	The user-defined name for the servlet context to be used within the domain.
`-virtualpath <path>`	Bind the newly created servlet context to this virtual path.
`-recreate`	If a context with this name already exists, delete it before adding an empty context with this name. This destroys any servlets currently associated with this context.
`-properties <prop_groups>`	List of property groups to use as the defaults for this service. Specify the name-value pairs in the same way as in the setgroup command.
`-docroot <location>`	All of the servle static pages are located in this directory in the server machine’s filesystem.

Table 1–40 createcontext Command Options

Option	Description
`-stateless`	All servlets in this context are stateless. Contexts declared to be stateless can only contain servlets that are stateless and never try to access the HTTPSession object.

The following example creates a servlet context within the domain "`ScottRoot`" of the name "`ScottContext`". All of the servlets should be loaded into the "`/private/scott/html`" directory. To access the servlets in this directory, use the virtual path of "`/SCOTT`".

```
createcontext -virtualpath /SCOTT -docroot /private/scott/html
                /ScottRoot ScottContext
```

destroycontext

Removes the servlet context, its information, and all contained servlets from the domain.

Syntax

```
destroycontext <context_name>
```

Argument Summary

Table 1–41 summarizes the `destroycontext` command-line arguments:

Table 1–41 destroycontext Command Options

Option	Description
<context_name>	The servlet context name, which was used on the createcontext command.

The following example remove the "`ScottContext`" servlet context. In addition, all servlets contained in the server’s filesystem directory "`/private/scott/html`" are all deleted, and the virtual path "`/SCOTT`" is removed.

```
destroycontext ScottContext
```

rmerrorpage

Remove the error code associated with the servlet context. This only removes the error code: it does not remove the servlet associated with the error code.

Syntax

```
rmerrorpage -error <errcode> <context_name>
```

Argument Summary

Table 1–42 summarizes the `rmerrorpage` command-line arguments:

Table 1–42 rmerrorpage Command Options

Option	Description
`<context_name>`	The name of the servlet context.
`-error <errcode>`	The error code that the error page is associated with. Deletes both the error code and the page that it is associated with.

The following command removes the previously defined error code 401 from the default servlet context. The servlet associated with this code may still exist.

```
rmerrorpage -error 401 /webdomains/context/default
```

Servlet Management

Once you have created the correct directory structure to contain your servlet, you can copy these into the server's filesystem. Then, you must publish these servlets in order for the client to invoke them.

publishservlet

Publish the servlet within the named servlet context.

Syntax

```
publishservlet [options] <context_name> <servlet_name> <class_name>
    [-virtualpath <path>]
    [-stateless]
    [-reuse]
    [-properties props]
```

Argument Summary

Table 1–43 summarizes the `publishservlet` command-line arguments:

Table 1–43 publishservlet Command Options

Option	Description
`<context_name>`	The name of the servlet context.
`<servlet_name>`	The name assigned to this servlet in the named_servlets directory to be published within this context. This name is used to refer to the class published with this command.
`<class_name>`	The name of the class implementing the HttpServlet interface.
`-virtualpath <path>`	Bind this servlet to this virtual path.
`-stateless`	This servlet is stateless and can not access the HTTPSession object.
`-reuse`	Add the virtual path to an existing servlet without republishing the servlet.
`-properties <prop_groups>`	List of property groups to use as the defaults for this service. Specify the name-value pairs in the same way as in the setgroup command.

The following command publishes the default context for the `HelloWorld` example that was loaded in SCOTT's schema:

```
publishservlet -virtualpath /hello /websdomains/contexts/default \
         helloServlet SCOTT:HelloWorld
```

unpublishservlet

Removes the servlet from the context as well as any existing virtual path for the servlet.

Syntax

```
unpublishservlet <context_name> <servlet_name>
```

Argument Summary

Table 1–44 summarizes the `unpublishservlet` command-line arguments:

Table 1–44 unpublishservlet Command Options

Option	Description
`<context_name>`	The name of the servlet context.

Table 1–44 unpublishservlet Command Options (Cont.)

Option	Description
`<servlet_name>`	The name of the servlet to be published within this context.

The following example unpublishes the `HelloWorld` servlet:

```
unpublishservlet /websdomains/contexts/default helloServlet
```

JavaServer Pages Management

Commands for publishing JavaServer Pages.These commands assume that the JSP definition is available as a resource on the server.

publishjsp

Translation, compilation, hotloading (if enabled), and publishing all occur automatically with the `publishjsp` command. This tool translates and publishes the JavaServer Pages in the designated servlet context. This command compiles the JavaServer Page into a servlet, which is stored in `jspresource`, and maintains the dependency between the `jspresource` and the generated class.

Run `publishjsp` after you have loaded a `.jsp` (or `.sqljsp`) file into Oracle8*i* as a resource schema object.

Syntax

```
publishjsp [options] <jsp_resource>
        -virtualpath <path>
        -schema <schema>
        -context <context>
        [-stateless]
        [-servletName <servlet_name>]
        [-packageName <pkg_name>]
        [-hotload]
        [-verbose]
        [-resolver <resolver>]
        [-extend <class>]
        [-implement <interface>]
```

Argument Summary

Table 1–45 summarizes the `publishjsp` command-line arguments:

Table 1–45 publishjsp Command Options

Option	Description
`<jsp_resource>`	The file *`name`*`.jsp` (or *`name`*`.sqljsp` for a SQLJ JSP page) is the JSP page resource schema object that you loaded with `loadjava` and is the only required parameter, along with any relevant schema *`path`* information.
`-virtualpath <path>`	Specify an alternative servlet path for the JSP page; otherwise, the servlet path is simply the specified `.jsp` file name itself along with any specified schema path. By default, `path/name.jsp` becomes the servlet path.
`-stateless`	The JSP page is to be stateless—the JSP page should not have access to the HttpSession object during execution. This flag is used for mod_ose optimization.
`-schema <schema>`	Specify the schema where the JSP page resource schema object is located, if it is not in the same schema you logged in to through `sess_sh`. This schema must be accessible from your `sess_sh` login schema. The `publishjsp` command does not offer a way to specify passwords.
`-servletName <servlet_name>`	Specify an alternative servlet name (in OSE named_servlets) for the JSP page. By default, the servlet name is the base name of the `.jsp` file along with any path you specified.
`-packageName <pkg_name>`	Specify a package name for the generated page implementation class. By default, it is the path specification for the `.jsp` file when you run `publishjsp`. This option affects where schema objects are placed in the schema, but does not affect the servlet path of the JSP page.
`-context <context>`	Specify a servlet context in the Oracle Servlet Engine. The context path of this servlet context becomes part of the URL used to invoke the page. The OSE default context, `/webdomains/contexts/default`, is `"/"`.
`-hotload`	Enable and perform hotloading. See the discussion on hotload following this table for more information.
`-verbose`	Report the translation steps during execution.
`-resolver <resolver>`	Specify an alternative Java class resolver. See the resolver discussion in the "loadjava" on page 1-7 for more information.
`-extend <class>`	Specify a Java class that the generated page implementation class extends.

Table 1–45 publishjsp Command Options (Cont.)

Option	Description
`-implement <interface>`	Specify a Java interface that the generated page implementation class implements.

Argument Details

hotload

Enable this flag to enable and perform hotloading. This results in the following steps being performed by the `publishjsp` command:

1. Static output is written to a resource schema object instead of to the page implementation class schema object.
2. A `main`() method and a hotloading method are implemented in the generated page implementation class to allow hotloading.
3. The `main`() method is executed to perform hotloading.
4. To use -hotload, you must have permission for the JServer hotloader. This can be granted as follows (from SQL*Plus, for the SCOTT schema, for example):

```
dbms_java.grant_permission('SCOTT',
    'SYS:oracle.aurora.security.JServerPermission', 'HotLoader', null);
```

The following example publishes the `Foo.jsp` into the default servlet context path of "`/`" and the default servlet path of "`dir1/Foo.jsp`".

```
publishjsp -schema SCOTT dir1/Foo.jsp
```

After this command, `Foo.jsp` can be invoked as follows:

`http://`*`host`*`[:`*`port`*`]/dir1/Foo.jsp`

Access it dynamically from another JSP page in the application, suppose a page published as `dir1/Bar.jsp`, as follows (using page-relative syntax and then application-relative syntax):

```
<jsp:include page="Foo.jsp" flush="true" />
```

unpublishjsp

Removes a JavaServer Page from the JNDI namespace. This does not remove the page implementation class schema object from the database. You do not need to specify a servlet name unless you specified one when you ran `publishjsp`.

Generally, the only required input is the servlet path, which is also known as the "virtual path".

Syntax

```
unpublishjsp [options] <servlet_path>
        [-servlet <servlet_name>]
        [-context <context>]
        [-showVersion]
        [-usage]
        [-verbose]
```

Argument Summary

Table 1–46 summarizes the `unpublishjsp` command-line arguments:

Table 1–46 unpublishjsp Command Options

Option	Description
`<servlet_path>`	Specify the servlet path for the JSP page.
`-servlet <servlet_name>`	Specify the servlet name for the JSP page. By default, the servlet name is the base name of the `.jsp` file along with any path you specified.
`-context <context>`	Specify a servlet context in the Oracle Servlet Engine. The OSE default context path is "/".
`-showVersion`	Display the OracleJSP version number
`-usage`	Display a option list
`-verbose`	Report the translation steps as it executes

The following example unpublishes the page that was published in the `publishjsp` section:

```
unpublishjsp dir1/Foo.jsp
```

Export Commands

Exports the structure of a web domain and its configuration file for the mod8i proxy.

The export utility works in two stages:

1. Generates, in XML format, the structure of a webdomain or contexts within a domain.
2. Optionally, apply transformations to the XML to produce configuration files specific to a the mod8i proxy.

exportwebdomain

This command creates the configuration files required for Apache's mod8i and for others. The default output is in an XML format.

Syntax

```
exportwebdomain [options] <domain_name>
    [-context <context>]
    [-netservice <name>]
    [-format <fmt>]
    [-nodefault]
    [-nodocs]
```

Argument Summary

Table 1–47 summarizes the `exportwebdomain` command-line arguments:

Table 1–47 exportwebdomain Command Options

Option	Description
`<domain_name>`	The name of the web domain that you want converted.
`-context <context>`	The name of the context to support. If not specified, all contexts within the domain are exported.
`-netservice <name>`	The name of the service defined in the Net8 initialization file (tnsnames.ora file).
`-format <fmt>`	The XSLT transformation defined in `<fmt>.xml` is used in the transformation of the domain. Use `-format apache` for mod_OSE configuration. Use `-format iis` for Microsoft Internet Service configuation. The `<fmt>.xml` files must be loaded as resources in the server under "`oracle/aurora/mts/http/admin/formats`".
`-nodefault`	Do not map the default context, unless indicated by the -context option.
`-nodocs`	Do not forward URLs mapped into doc_root to the Servlet engine. This assumes that such static pages will be served directly by the external webserver.

The following example exports the configuration that exists within the `/webdomain` domain to the "`/tmp/ApacheConfig`" file. The format is defined in the `Apache.xml` file and the Net8 connect string service name is `apache_connect`.

```
exportwebdomain -format Apache -netservice apache_connect \
            /webdomain & > /tmp/ApacheConfig
```

Security Management

In HTTP Security, access to a protected resource is composed of two parts, authentication and authorization. Authentication validates submitted credentials, which establishes that a user is known and validated by the system. Authorization determines whether an authorized user is allowed to perform the requested action.

There are four stages involved in establishing these security measures:

1. Declare the known principals of a service
2. Declare resources as being protected, and how they are to be protected
3. Declare the permissions of principals within the servlet context
4. Declare a security servlet for a servlet context

Without any one of these steps, security will either be non-existant or it will not allow any access to protected resources. These steps ensure that enough information has been declared, so that HTTP Security can successfully protect web resources.

Declaring Principals

Principal declarations are held in a "realm". A realm is made up of users and groups. The more generic term for either a user or a group is principal. When either entity can be used in a situation, the term principal should be used. Realm definitions exist within the scope of a web service. That is, all servlet contexts within a web service can use the same pools of principals for security.

- Principals have names by which they are identified within the system. Typically, this identification takes the form of a passphrase, but it is not always limited to this construct. Principals will have permissions declared for them, and may inherit any permissions that exist for groups of which they may be a member.
- Users have all of the same properties as principals

- Groups have all of the same properties as principals, as well as the property that other principals can be declared as being a member of a group. Thus, they inherit any permissions that exist for that group.
- Realms define sets of principals. There may be multiple realms within a single web service. The realm and its implementation are core to all of HTTP Security. A realm is the source of the following:
    - the valid set of principals
    - the types of principals that are handed to the server

  Since the realm is the source of all principals, it plays a key role in what types of credentials are to be used to identify principals, aiding the principals in managing the credentials themselves or can defer to whatever entity that does have them, and establishing the relationships among all principals within it.

By default, there are three implementations of realms for HTTP Security. They are identified by their types:

- JNDI—Stores all information in JNDI entries in the namespace
- DBUSER—Defers to local user and role definitions within the database itself
- RDBMS—Stores all principals and their relationships in database tables

These type names are shortcuts to use when declaring which realm class name to use in the JNDI entry that will be used to instantiate the realm.

Predefined Realms

The DBUSER realm derives all principal definitions from the users and roles defined within the database itself. No principal management is allowed through any of the security commands for this type of realm. The database, not the security tools, manages principal creation, deletion, and role membership. Since all instances of DBUSER realms utilize the same source for principal definition, all instances will essentially be equivalent. When referring to principals with a DBUSER realm, no case translations are performed. When the database entity was created, unless the case was explicitly expressed, the name will be all uppercase. For example SYS and PUBLIC must always be referred to in all uppercase. If a user were created, as follows, the username would exactly be '`joe`'.

```
create user "joe" identified by welcome;
```

This is especially important when it comes to supplying usernames and passwords from the browser.

Realm Management Overview

To create a RDBMS realm:

```
realm publish -w /myService -add testRealm -type RDBMS
```

Note: For JNDI and DBUSER, use those titles as the type argument.

To remove a realm:

```
realm publish -w /myService -remove testRealm
```

Note: It is by design of the system and its use of JNDI that realm declarations reside in the JNDI namespace. Deploying customized realms requires customizing the namespace entry.

To publish a custom realm:

```
realm publish -w /myService -add testRealm -classname foo.bar.MyRealm
```

Managing Principals Overview

Not all realms support the editing of principals. For example, DBUSER type realms do not support any principal manipulation.

To create a user:

```
realm user -w /myService -realm testRealm1 -add user1 -p upswd1
```

To create a group:

```
realm group -w /myService -realm testRealm1 -add group1 -p gpswd1
```

Note: In either of the above commands, if the password is left blank, the principal name is used instead.

To delete a user:

```
realm user -w /myService -realm testRealm1 -remove user1
```

To delete a group:

```
realm group -w /myService -realm testRealm1 -remove group1
```

To list users of a realm:

```
realm user -w /myService -realm testRealm1
```

To list groups of a realm:

```
realm group -w /myService -realm testRealm1
```

To add a principal to a group:

```
realm parent -w /myService -realm testRealm -group group1 -add user1
```

To remove a principal from a group:

```
realm parent -w /myService -realm testRealm -group group1 -remove user1
```

To list principals within a group:

```
realm parent -w /myService -realm testRealm -group group1
```

To query which groups a principal is member:

```
realm parent -w /myService -realm testRealm -q user1
```

Note: All realms do not support querying the principal group members.

Resource Protection Overview

In Aurora HTTP Security, resource protection is local to the servlet context. To declare a resource as protected, two pieces of information must be supplied. That information is embodied in a protection scheme. A scheme is of the form:

```
<authType>:<realmName>
```

Currently, there are only two valid authentication types, although these can be extended through JNDI namespace entries:

- Basic—typical base64 encoding, which is not secure
- Digest—both parties keep the password to themselves and pass highly encrypted codes, which are salted with situation specific values, such as timestamp, URL being requested, a secret key, and IP of the requester.

Note: Form based and SSL are currently not supported, though they may appear in a later update and can be plugged in through namespace entries.

You can declare resources to not be protected, which is useful when the servlet context root is to be protected. The problem is that when the root is protected, so are the error pages since they are part of the tree. In order to prompt for authentication, an error page is handed out. If that error page is protected, cycles develop and the desired behavior is not observed. Instead, explicitly declare the error pages as unprotected by using a protection scheme of `<NONE>`.

The path that describes what should be protected is local to the servlet context. Internally, that path is "normalized" to enable stable, predictable patterns for matching. This may cause the internal representation to differ from the original path used to create the protection scheme. HTTP Security uses the longest, most exact match possible when trying to apply the protection rules.

Protecting paths to resources with protection schemes:

```
realm map -s /myService/contexts/myContext -a /doc/index.html \
        -scheme basic:testRealm1
realm map -s /myService/contexts/myContext -a /doc -scheme basic:testRealm2
realm map -s /myService/contexts/myContext -a /doc/* -scheme basic:testRealm3
```

With the above declarations, here is how paths would be matched to realms:

```
/doc/index.html -> testRealm1
/doc -> testRealm2
/doc/ -> testRealm2
/doc/index -> testRealm3
/doc/foo -> testRealm3
```

To remove the protection of a path:

```
realm map -s /myService/contexts/myContext -r /doc/index.html
```

To list all protected paths within a servlet context:

```
realm map -s /myService/contexts/myContext
```

To explicitly declare a path not to be protected:

```
realm map -s /myService/contexts/myContext -a /system/* -scheme <NONE>
```

To list all protected paths within a servlet context:

```
realm map -s /myService/contexts/myContext
```

Declaring Permissions

Permissions are the most involved of all HTTP Security declarations. They tie web service scoped entities with servlet context scoped entities and they reside in the servlet context.

A permission declaration consists of several pieces:

- web service root
- realm within specified web service
- servlet context within specified web service
- principal within specified realm
- path to which the permission is to apply
- whether or not the permission being granted or denied
- HTTP actions being assigned

Given all of the pieces that are being tied into one permission declaration, it is easy to see why these are the most complicated declarations. HTTP Security permissions concern only those HTTP actions that are valid for HTTP requests, as follows:

  - GET
  - POST
  - PUT
  - DELETE
  - HEAD
  - TRACE
  - OPTIONS

To declare a granted permission on /foo/index.html for user1 for GET and POST:

```
realm perm -w /myService -realm testRealm1 -s /myService/contexts/myContext \
              -n user1 -u /foo/index.html + get,post
```

To declare a denied permission on `/foo/*` for `user1` for `PUT` and `DELETE`:

```
realm perm -w /myService -realm testRealm1 -s /myService/contexts/myContext \
            -n user1 -u /foo/* - put,delete
```

To remove granted permissions on `/foo/index.html` for `user1`:

```
realm perm -w /myService -realm testRealm1 -s /myService/contexts/myContext \
              -n user1 -u /foo/index.html +
```

To list all permissions for a user:

```
realm perm -w /myService -realm testRealm1 -s /myService/contexts/myContext \
             -n user1
```

Declaring A Security Servlet

All HTTP Security is declared through JNDI namespace entries. This is also true for the servlet that does the enforcing of security. In the servlet context, if there is a `PrivilegedServlet` named `httpSecurity`, that servlet is added as the first pre-filter for all requests within that servlet context.

As with all JNDI namespace entries so far in HTTP Security, if customization is desired, the namespace entry can specify any custom servlet, if it implements the `PrivilegedServlet` interface. If this servlet is customized, it can handle security any way it chooses, since its main responsibility is to do one of the following for (`HttpRequest.PrivilegedAccess, HttpRequest, HttpResponse`):

- Raise an `AccessControlException` during its service if there is any security violation
- Allow the the request

Once authentication and authorization have taken place, it is also the responsibility of the servlet to set specific authenticated principal values on the request itself. This is the user information that can be retrieved from the request by any executing servlet.

To create a security servlet:

```
realm secure -s /myService/contexts/myContext
```

Removing the security servlet removes all security enforcement in a servlet context. If the entry is missing, the webserver continues execution with no security enforcement.

To remove a security servlet:

```
rm /myService/contexts/myContext/httpSecurity
```

Note: The servlet is not published in `namedServlets`, but within the `servletContext` itself.

realm

The realm command suite, listed below, manages all realm constituents.

■ list	■ echo	■ secure
■ map	■ publish	■ user
■ group	■ parent	■ perm

list

Lists the realms declared for the given web service.

The syntax is as follows:

```
realm list -w <webServiceRoot>
```

where the <*webServiceRoot*> is the web service to list.

```
realm list -w /webservice
```

echo

Can be used to suppress the results of subsequent realm commands.

The syntax is as follows:

```
realm echo [0 | 1]
```

- 0: Do not print any results of subsequent operations
- 1: Print results of subsequent operations. This is the default.

```
realm echo 0
```

secure

Used to setup the default security servlet for a given servlet context.

The syntax is as follows:

```
realm secure -s <servletContextPath>
```

where `servletContextPath` is the servlet context to operate upon.

map

Used to map paths local to the given servlet context to protection schemes. This is what declares a resource to be protected.

The syntax is as follows:

```
realm map -s servletContextPath [-(a[dd]|r[emove]) <path> -scheme auth:realm]
```

Table 1–48 map Option Description

Option	Description
`-s servletContextPath`	The servlet context to operate upon. If nothing else is supplied, this will list all URL-Scheme mappings for the given servlet context.
`-add | -a <path>`	The path for which a mapping will be added, or edited if the mapping already exists.
`-remove | -r <path>`	The path for which a mapping will be removed.
`-scheme auth:realm`	The protection scheme to use for the given mapping.

publish

User to publish and remove various types of realms within a given web service. If the realm already exists, information about that pre-existing realm is output.

The syntax is as follows:

```
realm publish -w <webserviceRoot> [-(a[dd]|r[emove]) <realmName>
       [-type <realmType>]]
```

Table 1–49 publish Option Summary

Option	Description
`-w <webserviceRoot>`	The web service to operate upon.
`-add | -a <realmName>`	The name to use for creating the realm within the web service.
`-remove | -r <realmName>`	The name of the realm to remove from the web service.
`-type <realmType>`	Specifies the type of realm to publish. One of RDBMS, DBUSER, JNDI. If not specified, the default value is RDBMS.

user

Used to query what users exist within a realm, add users to a realm, remove users from a realm, edit the passwords of existing users.

The syntax is as follows:

```
realm user -w <webserviceRoot> -realm <realmName>
      [-(a[dd]|r[emove]) <userName> [-p <user> <password>]]
```

Table 1–50 user Option Summary

Option	Description
`-w <webserviceRoot>`	The web service to operate upon.
`-realm <realmName>`	The name of the realm to operate upon. If no other arguments are supplied, the names of all users within the given realm are output.
`-add | -a <userName>`	The name of the user to create within the realm.
`-remove | -r <userName>`	The name of the user to remove from the realm.
`-p <password>`	The password to be associated with the user. If not supplied, the user name is used instead. If the user already exists, the user's password is reset to this value.

group

Used to query what groups exist within a realm, add groups to a realm, remove groups from a realm, edit the passwords of existing groups.

The syntax is as follows:

```
realm group -w <webserviceRoot> -realm <realmName>
          [-(a[dd]|r[emove]) <groupName> [-p <group> <password>]]
```

Table 1–51 group Option Summary

Option	Description
`-w <webserviceRoot>`	The web service to operate upon.
`-realm <realmName>`	The name of the realm to operate upon. If no other arguments are supplied, the names of all users within the given realm are output.
`-add | -a <groupName>`	The name of the group to create within the realm.
`-remove | -r <groupName>`	The name of the group to remove from the realm.

Table 1–51 group Option Summary

Option	Description
`-p <password>`	The password to be associated with the group. If not supplied, the group name is used instead. If the group already exists, the group's password is reset to this value.

parent

Used to query and manage principal-group relationships.

The syntax is as follows:

```
realm parent -w webserviceRoot -realm realmName [-g[roup] groupName
      [-(a[dd]|r[emove]) principalName]] [-q[uery] principalName
```

Table 1–52 parent Option Summary

Option	Description
`-w webserviceRoot`	The web service to operate upon.
`-realm realmName`	The name of the realm to operate upon.
`-group groupName`	The group to operate upon. If no other arguments are supplied, all members of this group are output, if the given realm supports such an action.
`-add principalName`	The name of the principal to add to this group. Some realms may not support this action or may disallow this operation if it detects a circularity in the group-principal membership chain.
`-remove principalName`	The name of the principal to remove from the group. Some realms may not support this action.
`-query principalName`	Print the names of all groups that have this principal as a member. Some realms may not support this action.

perm

Used to query and manage permissions for principals.

The syntax is as follows:

```
realm perm -w webserviceRoot -realm realmName -s servletContextPath
     -n[ame] principalName [-p[ath] path (+|-) permList]
```

Table 1–53 perm Option Summary

Option	Description
`-w webserviceRoot`	The web service to operate upon.
`-realm realmName`	The name of the realm to operate upon.
`-s servletContextPath`	The servlet context to operate upon.
`-name principalName`	The name of the principal for which permission operations will be performed. If no other arguments are supplied, then print out all permissions for this principal.
`-path path`	The path to be used when applying the permission.
`+ permList`	The list of HTTP actions to grant to the user. If permList is not given, then clear all granted permissions for the user. This is not an additive operation, any existing granted permissions for the user are completely over-written by the new list.
`- permList`	The list of HTTP actions to deny to the user. If permList is not given, then clear all denied permissions for the user. This is not an additive operation, any existing denied permissions for the user are completely over-written by the new list.

Note: The `permList` is a comma separated list of HTTP actions with no spaces. For Example: `get,post,trace`

Enterprise JavaBean Tools

Instead of `loadjava` and `publish`, Enterprise JavaBean developers use the `deployejb` tool, which performs equivalent operations, as well as generating and compiling infrastructure code for the EJB. The `ejbdescriptor` tool is a utility for translating between the text and serialized object forms of EJB deployment descriptors.

deployejb

From a deployment descriptor and a JAR containing interfaces and classes, the `deployejb` tool makes an EJB implementation ready for test or production clients to invoke. `deployejb` converts the text descriptor to a serialized object, generates and compiles classes that effect client-bean communication, loads compiled classes into the database, and publishes the bean's home interface name in the session namespace so clients can look it up with JNDI. The BeanHomeName must refer to a PublishingContext for which the `deployejb` invoker has the write right; see "publish" on page 1-21 for the rights required to publish.

Before deploying, verify that you add the appropriate JDK JAR, library, and binary information in the following environment variables:

Environment Variable	**Addition Required**
`JAVA_HOME`	Set to the location where the JDK is installed.
`CLASSPATH`	Include the appropriate JDK JAR file in your CLASSPATH, as follows: ■ For JDK 1.1, include `$JAVA_HOME/lib/classes.zip` ■ For JDK 1.2, include the JDK's `tools.jar` and `dt.jar` file Also, include the remote and home interface files and the JAR generated by `deployejb`.
`PATH`	Add the JDK binary path: `$JAVA_HOME/bin`
`LD_LIBRARY_PATH`	Add the JDK library path: `$JAVA_HOME/lib`

To specify a different encoding for multibyte support, modify the encoding element in the XML deployment descriptor heading. The `deployejb` tool recognizes the proper encoding from the header.

Syntax

```
deployejb {-user | -u} <username> {-password | -p} <password>
  {-service | -s} <serviceURL> -descriptor <file> -temp <work_dir> <beanjar>
  [-addclasspath <dirlist>]
  [-beanonly]
  [-credsFile <credentials>]
  [-describe | -d]
  [-generated <clientjar>]
  [-help | -h]
  [-iiop]
  [-keep]
  [-oracledescriptor <file>]
  [-republish]
  [-resolver "resolver_spec"]
  [-role <role>]
  [-ssl]
  [-useServiceName]
  [-verbose]
  [-version | -v]
```

Argument Summary

Table 1–54 summarizes the `deployejb` arguments.

Note: Any value provided within the argument options is case insensitive. All values are uppercased.

Table 1–54 deployejb Argument Summary

Argument	Description and Values
`-user`	Specifies the schema into which the EJB classes will be loaded.
`-password`	Specifies the password for `<username>`.
`-service`	URL identifying database in whose session namespace the EJB is to be published. The serviceURL has the form: `sess_iiop://<host>:<lport>:<sid>` `<host>` is the computer that hosts the target database; `<lport>` is the listener port configured to listen for session IIOP; `<sid>` is the database instance identifier. Example: `sess_iiop://localhost:2481:orcl` which matches the default installation on the invoker's machine.

Table 1–54 deployejb Argument Summary (Cont.)

Argument	Description and Values
`-credsFile`	Supply a text file with credentials instead of a username and password for the connect. You create this file by exporting a wallet into a text version.
`-descriptor`	Specifies the text file containing the EJB deployment descriptor.
`-temp`	Specifies a temporary directory to hold intermediate files `deployejb` creates. Unless you specify `-keep`, `deployejb` removes the files and the directory when it completes.
`<beanjar>`	Specifies the name of the JAR containing the bean interface and implementation files.
`-addclasspath`	Specifies directories containing interface and / or implementation dependency classes not contained in `<beanjar>`. Format of `<dirlist>` is the same as `javac`'s `CLASSPATH` argument. Required for `-beanonly`.
`-beanonly`	Skips generation of interface files. Basically, this option enables you to reload the bean implementation if none of the interfaces have changed.
`-describe`	Summarizes the tool's operation.
`-generated`	Specifies the name of the output (generated) JAR file, which contains communication files bean clients need. If you do not specify, the output JAR file has the name of the input JAR file with `_generated` appended.
`-help`	Summarizes the tool's syntax.
`-iiop`	Connects to the target database with IIOP instead of the default session IIOP. Use this option when deploying to a database server that has been configured without session IIOP.
`-keep`	Do not remove the temporary files generated by the tool. This option may be useful for debugging because it provides access to the source files `deployejb` generates.
`-oracledescriptor`	Specifies the text file containing the Oracle-specific deployment descriptor.
`-republish`	Replaces the published BeanHomeName attributes if the BeanHomeName has already been published, otherwise publishes it.

Table 1–54 deployejb Argument Summary (Cont.)

Argument	Description and Values
`-resolver`	Specifies an explicit resolver spec, which is bound to the newly loaded classes. If -resolver is not specified, the default resolver spec, which includes current user's schema and PUBLIC, is used. For more information, see the discussion on -resolve and -resolver in "loadjava" on page 1-7.
`-role`	Specifies role to assume when connecting to the database; no default.
`-ssl`	Connects to the database with SSL authentication and encryption.
`-useServiceName`	If you are using a service name instead of an SID in the URL, you must specify this flag. Otherwise, the tool assumes the last string in the URL is the SID.
`-verbose`	Emits detailed status information while running.
`-version`	Shows the tool's version.

Argument Details

addclasspath

`deployejb` needs the classes the home and remote interfaces depend on and the classes the bean implementation depends on. These dependency classes can either be included in the `<beanjar>` file or directories containing them or can be specified in the `-addclasspath` argument. The first approach is less prone to error, the second can substantially reduce `deployejb`'s run time. If you use `-addclasspath`, then you must ensure that the classes have been loaded before you run a client that activates the EJB.

Here is a `deployejb` example.

Basic invocation specifying the name of the generated client JAR file:

```
deployejb -user SCOTT -password TIGER -service sess_iiop://dbserver:2481:orcl \
  -descriptor myBeanDescriptor.xml -temp /tmp/ejb \
  -generated myBeanClient.jar myBean.jar
```

ejbdescriptor

Each EJB implementation includes a serialized Java object known as a deployment descriptor. The values in a deployment descriptor are not readable by people, yet people must create them and might sometimes have to read them. The `ejbdescriptor` tool transforms a serialized deployment descriptor to text and converse. Developers are most likely to use `ejbdescriptor` to extract the deployment descriptor data from an EJB developed for a non-Oracle environment. The `deployejb` tool calls `ejbdescriptor` to build a deployment descriptor from the text file you specify in the `-descriptor` argument.

Syntax

```
ejbdescriptor [-options] <infile> <outfile>
  [-parse]
  [-parsexml]
  [-dump
  [-dumpxml]
  [-encoding]
```

Argument Summary

Table 1–55 describes the `ejbdescriptor` arguments.

Table 1–55 ejbdescriptor Argument Summary

Option	Description
`-parse`	Creates serialized deployment descriptor `<outfile>` from a Release 8.1.6 and previous .ejb text deployment descriptor specified in `<infile>`.
`-parsexml`	Creates the Release 8.1.6 .ejb text deployment descriptor `<outfile>` from and XML deployment descriptor specified in `<infile>`.
`-dump`	Creates a Release 8.1.6 .ejb deployment descriptor text file `<outfile>` from serialized deployment descriptor `<infile>`.
`-dumpxml`	Creates the Release 8.1.7 XML deployment descriptor file `<outfile>` from a Release 8.1.6 text deployment descriptor `<infile>`.
`-encoding`	Identifies the source file encoding for the compiler, overriding the matching value, if any, in the `JAVA$OPTIONS` table. Values are the same as for the `javac -encoding` option. If you do not specify an encoding on the command line or in a `JAVA$OPTIONS` table, the encoding is assumed to be `latin1`. The `-encoding` option is relevant only when loading a source file.
`infile`	Name of the file to parse or read. The default is standard in. The conventional suffix for a deployment descriptor file is .ejb or .xml; for a serialized descriptor it is .ser.

Table 1–55 ejbdescriptor Argument Summary (Cont.)

Option	Description
`outfile`	Name of file to `dump` or write. The default is standard out. The conventional suffix for a deployment descriptor file is .ejb or .xml; for a serialized descriptor it is .ser.

Here are examples of the `ejbdescriptor` tool.

Create a Release 8.1.7 XML deployment descriptor from a Release 8.1.6 `.ejb` deployment descriptor:

```
ejbdescriptor -dumpxml beandescriptor.ejb beandescriptor.xml
```

Create a Release 8.1.6 deployment descriptor from an XML deployment descriptor:

```
ejbdescriptor -parsexml beandescriptor.xml beandescriptor.ser
```

Create a text file representation of a Release 8.1.6 deployment descriptor:

```
ejbdescriptor -dump beandescriptor.ser beandescriptor.ejb
```

Create a serialized deployment descriptor from a Release 8.1.6 deployment descriptor file:

```
ejbdescriptor -parse beandescriptor.ejb beandescriptor.ser
```

Display the contents of a Release 8.1.6 deployment descriptor:

```
ejbdescriptor -dump beandescriptor.ser
```

VisiBroker™ for Java Tools

JServer incorporates the Inprise (Visigenic) Caffeine tools that allow you to code object interfaces directly in Java and generate the infrastructure necessary to support distributed object invocation. These tools include:

- *`java2rmi_iiop`* generates the infrastructure EJB requires to call other remote objects. `java2rmi_iiop` is an extension of the Inprise `java2iiop` tool.
- *`java2idl`* compiles Java interfaces to IDL code, for cases where IDL is required.

The `idl2java`, `java2idl`, and `java2iiop` tools developed by Inprise for their VisiBroker for Java product (release 3.4) are distributed with Oracle8*i*. The Oracle8*i* JServer CD contains the documentation for these tools; the documentation can also be viewed or downloaded from `http://www.inprise.com`. Because the Oracle8*i* run-time environment differs somewhat from the VisiBroker environment, some VisiBroker tool options might not work in Oracle8*i* JServer as they are described in the VisiBroker documentation.

Native Compilation Tools

The Java language was designed for a platform-independent, secure development model. To accomplish these goals, some execution performance was sacrificed. Translating Java bytecodes into machine instructions degrades performance. To regain some of the performance loss, you may choose to natively compile certain classes. For example, you may decide to natively compile code with CPU intensive classes.

Without native compilation, the Java code you load to the server is interpreted and the underlying core classes upon which your code relies (`java.lang.*`) are natively compiled.

Native compilation provides a speed increase ranging from two to ten times the speed of the bytecode interpretation. The exact speed increase is dependent on several factors, including:

- use of numerics
- degree of polymorphic message sends
- use of direct field access, as opposed to accessor methods
- amount of Array accessing
- casts

Because Java bytecodes were designed to be compact, natively compiled code can be considerably larger than the original bytecode. However, because the native code is stored in a shared library, it is shared among all users of the database.

Most JVMs use Just-In-Time compilers that convert the Java bytecodes to native machine instructions when methods are invoked. The JServer Accelerator uses an Ahead-Of-Time approach to recompiling the Java classes.

Native Compiler	Description
Just-In-Time	Provides the JVM the ability to translate the Java instructions just before needed by the JDK. The benefits depends on how accurately the native compiler anticipates code branches and the next instruction. If incorrect, no performance gain is realized.

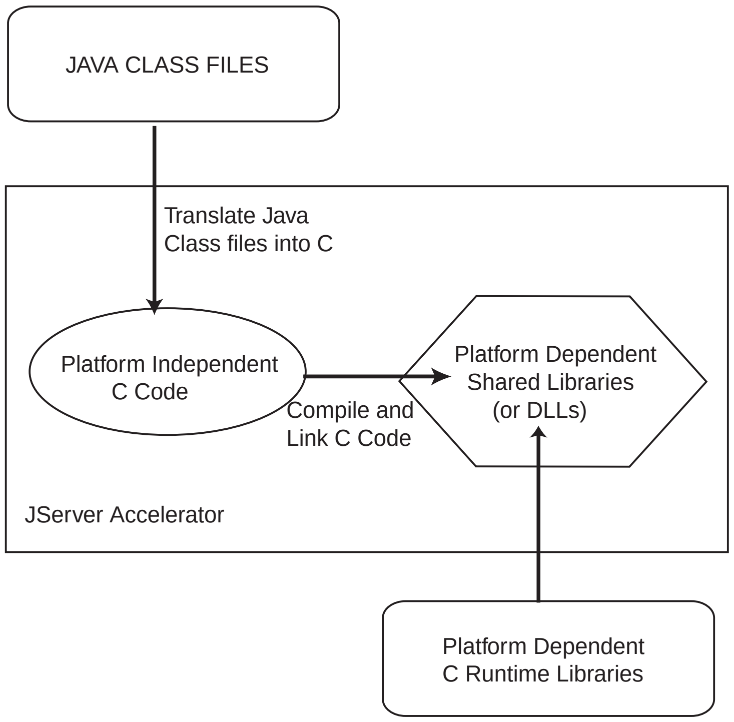
Native Compiler	Description
Ahead-Of-Time	The Jserver Accelerator natively compiles all Java code within a JAR file into native shared libraries, which are organized by Java package, before execution time. At runtime, JServer Accelerator checks if a Java package has been natively compiled; and if so, uses the machine code library instead of interpreting the deployed Java code.

This static compilation approach provides a large, consistent performance gain, regardless of the number of users or the code paths they traverse on the server. After compilation, the tool loads the statically compiled libraries into JServer, which are then shared between users, processes, and sessions.

JServer Accelerator Overview

Most Ahead-Of-Time native compilers compile directly into a platform-dependent language. For portability requirements, this was not feasible. As shown in Figure 1–1, the JServer Accelerator translates the Java classes into a version of C that is platform-independent. This C code is compiled and linked to supply the final platform-dependent, natively compiled shared libraries or DLLs.

Figure 1–1 Native Compilation using JServer Accelerator


JAVA CLASS FILES
Translate Java Class files into C
Platform Independent C Code
Compile and Link C Code
Platform Dependent Shared Libraries (or DLLs)
JServer Accelerator
Platform Dependent C Runtime Libraries

Given a JAR file, the JServer Accelerator performs the following:

1. The classes, loaded in the database, are verified.
2. The Java bytecodes for these classes are retrieved from the database and stored in a project directory where the JServer Accelerator was invoked.
3. The Java bytecodes are translated to C.
4. The C code is compiled and linked with the C compiler for your platform.

   JServer Accelerator translates, compiles, and links the retrieved classes on the client. For this reason, you must natively compile on the intended platform environment that this application will be deployed to. The result is a single deployment JAR file for all classes within the project.
5. The resulting shared library is loaded into the `$ORACLE_HOME/javavm/admin` directory.

Note: The JServer Accelerator natively compiled libraries can only be used within JServer. Also, these libraries can only be used within the same version of JServer that it was produced in. If you want your application to be natively compiled on subsequent releases, you must recompile these classes. That is, native recompilation of existing libraries will not be performed automatically by any upgrade process.

JServer Core Java Class Libraries

All core Java class libraries and Oracle-provided Java code within JServer is natively compiled for greater execution speed. Java classes exist as shared libraries in `$ORACLE_HOME/javavm/admin`, where each shared library corresponds to a Java package. For example, `orajox8java_lang.so` on Solaris and `orajox8java_lang.dll` on Windows NT hold `java.lang` classes. Specifics of packaging and naming can vary by platform. The Aurora JVM uses natively compiled Java files internally and opens them, as necessary, at runtime.

Natively Compiling Java Application Class Libraries

The JServer Accelerator can be used by Java application products that need an performance increase and are deployed on JServer. The JServer Accelerator command-line tool, `ncomp`, natively compiles your code and loads it in JServer. However, in order to use `ncomp`, you must first provide some initial setup.

Installation Requirements

You must install the following before invoking JServer Accelerator:

1. Install a C compiler for the intended platform on the machine you are running `ncomp`.
2. Verify that the correct compiler and linker commands are referenced within the `System*.properties` file located in the `$ORACLE_HOME/javavm/jahome` directory. Since the compiler and linker information is platform-specific, the configuration for these items is detailed in the README for your platform.
3. Add the appropriate JDK JAR files, library, and binary information in the following environment variables:

Environment Variables	Addition Required
`JAVA_HOME`	Set to the location where your JDK is installed.
`CLASSPATH`	Include the appropriate JDK JAR files in your CLASSPATH as follows: ■ For JDK 1.1, include `$JAVA_HOME/lib/classes.zip`. ■ For JDK 1.2, include the `$JAVA_HOME/lib/tools.jar` and `$JAVA_HOME/lib/dt.jar` files.
`PATH`	Add the JDK binary path: `$JAVA_HOME/bin`
`LD_LIBRARY_PATH`	Add the JDK library path: `$JAVA_HOME/lib`.

4. Grant the user that executes `ncomp` the following role and security permissions:

> **Note:** DBA role contains both the `JAVA_DEPLOY` role and the `FilePermission` for all files under `$ORACLE_HOME`.

a. `JAVA_DEPLOY`: The user must be assigned to the `JAVA_DEPLOY` role in order to be able to deploy the shared libraries on the server, which both the `ncomp` and `deploync` utilities perform. For example, the role is assigned to DAVE, as follows:

```
SQL> GRANT JAVA_DEPLOY TO DAVE;
```

b. `FilePermission`: JServer Accelerator stores the shared libraries with the natively compiled code on the server. In order for JServer Accelerator to store these libraries, the user must be granted `FilePermission` for read and write access to directories and files under `$ORACLE_HOME` on the server. One method for granting `FilePermission` for all desired directories is to grant the user the JAVASYSPRIV role, as follows:

```
SQL> GRANT JAVASYSPRIV TO DAVE;
```

See the Security chapter in the *Oracle8i Java Developer's Guide* for more information JAVASYSPRIV and granting `FilePermission`.

ncomp

JServer Accelerator, implemented within the `ncomp` tool, natively compiles all classes within the specified JAR, ZIP, or list of classes. JServer Accelerator natively compiles these classes and places them into shared libraries according to their package. Note that these classes must first be loaded into the database.

If the classes are designated within a JAR file and have already been loaded in the database, you can natively compile your Java classes by executing the following:

```
ncomp -user SCOTT/TIGER myClasses.jar
```

Note: Because native compilation must compile and link all of your Java classes, this process may execute over the span of a few minutes or a few hours. The time involved depends on the number of classes to compile and the type of hardware on your machine.

There are options that allow you control over how the details of native compilation are handled.

Syntax

```
ncomp [ options ] <class_designation_file>
  -user | -u <username>/<password>[@<database_url>]
  [-load]
  [-projectDir | -d <project_directory>]
  [-force]
  [-lightweightDeployment]
  [-noDeploy]
  [-outputJarFile | -o <jar_filename>]
  [-thin]
  [-oci8]
  [-update]
  [-verbose]
```

Note: These options are demonstrated within the Scenarios described in "Native Compilation Usage Scenarios" on page 1-105.

Argument Summary

Table 1–56 summarizes the `ncomp` arguments. The *`<class_designation_file>`* can be a *`<file>.jar`*, *`<file>.zip`*, or *`<file>.classes`*.

Table 1–56 ncomp Argument Summary

Argument	Description and Values
`<file>.jar`	The full pathname and filename of a JAR file that contains the classes that are to be natively compiled. If you are executing in the directory where the JAR file exists and you do not specify the -projectDir option, you may give only the name of the JAR file.
`<file>.zip`	The full pathname and filename of a ZIP file that contains the classes that are to be natively compiled. If you are executing in the directory where the ZIP file exists and you do not specify the -projectDir option, you may give only the name of the ZIP file.
`<file>.classes`	The full pathname and filename of a classes file, which contains the list of classes to be natively compiled. If you are executing in the directory where the classes file exists and you do not specify the -projectDir option, you may give only the name of the classes file. See "Natively Compiling Specific Classes" on page 1-107 for a description of a classes file.
`-user | -u <username>/<password> [@<database>]`	Specifies a user, password, and database connect string; the files will be loaded into this database instance. The argument has the form `<`*`username`*`>/<`*`password`*`>[@<`*`database`*`>]`. If you specify the database URL on this option, you must specify it with OCI8 syntax. To provide a JDBC Thin database URL, use the -thin option.
`-force`	The native compilation is performed on all classes. Previously compiled classes are not passed over.
`-lightweightDeployment`	Provides an option for deploying shared libraries and native compilation information separately. This is useful if you need to preserve resources when deploying. See "lightweightDeployment" on page 1-104 for more information.
`-load`	Executes loadjava on the specified class designation file. You cannot use this option in combination with a <*file*>.classes file.
`-outputJarFile <jar_filename>`	All natively compiled classes output into a deployment JAR file. This option specifies the name of the deployment JAR file and its destination directory. If omitted, the ncomp tool names the output deployment JAR file the same name as the input <*file*> with "_depl.jar" appended as the suffix. If directory is not supplied, it stores the output JAR file into the project directory (denoted by -projectDir).

Table 1–56 ncomp Argument Summary

Argument	Description and Values
`-noDeploy`	Specifies that the native compilation results only in the output deployment JAR file, which is not deployed to the server. The resulting deployment JAR can be deployed to any server using the deploync tool.
`-thin`	The database URL that is provided on the -user option uses a JDBC Thin URL address for the database URL syntax.
`-oci8`	The database URL that is provided on the -user option uses an OCI8 URL address for the database URL syntax. However, if neither -oci8 or -thin are specified, the default assumes that you used an OCI8 database URL.
`-projectDir | -d <absolute_path>`	Specifies the full path for the project directory. If not specified, JServer Accelerator uses the directory that ncomp is invoked from as the project directory. This directory must exist; the tool will not create this directory for you. If it does not exist, the current directory is used.
`-update`	If you add more classes to a *<class_designation_file>* that has already been natively compiled, this flag informs JServer Accelerator to update the deployment JAR file with the new classes. Thus, JServer Accelerator compiles the new classes and adds them to the appropriate shared libraries. The deployment JAR file is updated.
`-verbose`	Output native compilation text with detail.

Argument Details

user

{-user | -u} <user>/<password>[@<database>]

The permissible forms of `@<database>` depend on whether you specify `-oci8` or `-thin`; `-oci8` is the default.

- `-oci8`: `@<database>` is optional; if you do not specify, then `ncomp` uses the user's default database. If specified, then `<database>` can be a TNS name or a Net8 name-value list.
- `-thin`: `@<database>` is required. The format is `<host>:<lport>:<SID>`.
  - `<host>` is the name of the machine running the database.
  - `<lport>` is the listener port that has been configured to listen for Net8 connections; in a default installation, it is 5521.

   – <SID> is the database instance identifier; in a default installation, it is ORCL.

lightweightDeployment

JServer Accelerator places compilation information and the compiled shared libraries in one JAR file, copies the shared libraries to $ORACLE_HOME/javavm/admin directory on the server, and deploys the compilation information to the server. If you want to place the shared libraries on the server yourself, you can do so through the lightweightDeployment option. The lightweightDeployment option enables you to do your deployment in two stages:

1. Natively compile your JAR file with -noDeploy and -lightweightDeployment options. This creates an deployment JAR file with only ncomp information, such as transitive closure information. The shared libraries are not saved within the deployment JAR file. Thus, the deployment JAR file is much smaller.
2. Deploy as follows:
   a. Copy all output shared libraries from the lib directory of the native compilation project directory to the server's $ORACLE_HOME/javavm/admin directory.

      Note: You need to have FilePermission to write to this directory. FilePermission is included in the DBA or JAVASYSPRIV roles.

   b. Deploy the lightweight deployment JAR file to the server using deploync.

Errors

Any errors that occur during native compilation are printed to the screen. Any errors that occur during deployment of your shared libraries to the server or during runtime can be viewed with the statusnc tool or by referring to the JACCELERATOR$DLL_ERRORS table.

If an error is caught while natively compiling the designated classes, JServer Accelerator denotes these errors, abandons work on the current package, and continues its compilation task on the next package. The native compilation continues for the rest of the packages. The package with the class that contained the error will not be natively compiled at all.

After fixing the problem with the class, you can choose to do one of the following:

- recompile the shared library
- reload the Java class into the database

If you choose to not recompile the classes, but to load the correct Java class into the database, the corrected class and all classes that are included in the resolution validation for that class—whether located within the same shared library or a different shared library—will be executed in interpreted mode. That is, the JVM will not run these classes natively. All the other natively compiled classes will continue to execute in native format. When you execute the statusnc command on the reloaded class or any of its referred classes, they will have an NEED_NCOMPING status message.

Possible errors for a Java class:

1. The Java class does not exist in the database. If you do not load the Java class into JServer, JServer Accelerator does not include the class in the shared library. The class is simply skipped.
2. The Java class is invalid; that is, one of its references may not be found.
3. Any Java class that is unresolved, JServer Accelerator will try to resolve it before natively compiling. However, if the class cannot be resolved, it is ignored by JServer Accelerator.

Possible errors for deployment of native compilation JAR file:

- The native compilation of your JAR file executes correctly, but the deployment fails. In this case, do not recompile the JAR file, but deploy the output natively compiled JAR file with the deploync command.

Native Compilation Usage Scenarios

The following scenarios demonstrate how each of the options for the `ncomp` tool can be used:

- Natively Compiling on Test Platform—Java Classes Already Loaded in the Database
- Natively Compiling Java Classes Not Loaded in the Database
- Clean Compile and Generate Output for Future Deployment
- Controlling Native Compilation Build Environment
- Natively Compiling Specific Classes

- Natively Compiling Packages That Are Fully or Partially Modified

Natively Compiling on Test Platform—Java Classes Already Loaded in the Database

If all classes are loaded into the database and you have completed your testing of the application, you can request JServer Accelerator to natively compile the tested classes. JServer Accelerator takes in a JAR, ZIP, or list of classes to determine the packages and classes to be involved in the native compilation. The JServer Accelerator then retrieves all of the designated classes from the server, natively compiles them into shared libraries—each library containing a single package of classes.

Assuming that the classes have already been loaded within the server, you execute the following command to natively compile all classes listed within a class designation file, such as the `pubProject.jar` file, as follows:

```
ncomp -user SCOTT/TIGER pubProject.jar
```

If you change any of the classes within the class designation file and ask for recompilation, JServer Accelerator recompiles only the packages that contain the changed classes. It will not recompile all packages.

Natively Compiling Java Classes Not Loaded in the Database

Once tested, you may wish to natively compile the designated classes on another host than the test machine. Once you transfer the designated class file to this platform, the classes in this file must be loaded into the database before native compilation can occur. The following loads the classes through `loadjava` and then executes native compilation for the class designation file—`pubProject.jar`:

```
ncomp -user SCOTT/TIGER@dbhost:5521:orcl -thin -load pubProject.jar
```

Clean Compile and Generate Output for Future Deployment

If you want all classes within a class designation file to be recompiled—regardless of whether they were previously natively compiled—you execute `ncomp` with the `-force` option. You might want to use the `-force` option to ensure that all classes are compiled resulting in a deployment JAR file that can be deployed to other JServer databases. You can specify the native compilation deployment JAR file with the `-outputJarFile` option. The following forces a recompilation of all Java classes within the class designation file—`pubProject.jar`—and creates a deployment JAR file with the name of `pubworks.jar`:

```
ncomp -user SCOTT/TIGER -force -outputJarFile pubworks.jar pubProject.jar
```

The deployment JAR file contains the shared libraries for your classes and installation classes specified to these shared libraries. It does not contain the original Java classes. In order to deploy the natively compiled deployment JAR file to any JServer (of the appropriate platform type), you must do the following:

1. Load the original Java classes into the destination server. In the previous example, the `pubProject.jar` file would be loaded into the database using the `loadjava` tool.
2. Deploy the natively compiled deployment JAR file with the JServer Accelerator deploync tool, which is described in deploync on page 1-108.

Controlling Native Compilation Build Environment

By default, the JServer Accelerator uses the directory where `ncomp` is executed as its build environment. The JServer Accelerator downloads several class files into this directory, and then uses this directory for the compilation and linking process.

If you do not want to have JServer Accelerator put any of its files into the current directory, create a working directory, and specify this working directory as the project directory with the `-projectDir` option. The following directs JServer Accelerator to use `/tmp/jaccel/pubComped` as the build directory. This directory must exist before specifying it within the -projectDir option. JServer Accelerator will not create this directory for you.

```
ncomp -user SCOTT/TIGER -projectDir /tmp/jaccel/pubComped pubProject.jar
```

Natively Compiling Specific Classes

You can specify one or more classes, which are to be natively compiled, within a text-based `<`*`file`*`>.classes` file. You use the following Java syntax to specify packages and/or individual classes within this file:

- To specify classes within one or more packages, as follows:

```
import COM.myDomain.myPackage.*;
import COM.myDomain.myPackage.mySubPackage.*;
```

Note: Java has no formal notion of a sub-package. You must specify each package independently.

- To specify an individual class, as follows:

```
import COM.myDomain.myPackage.myClass;
```

Once explicitly listed, you specify the name and location of this class designation file on the command line. Given the following pubworks.classes file:

```
import COM.myDomain.myPackage.*;
import COM.myDomain.hisPackage.hisSubPackage.*;
import COM.myDomain.herPackage.herClass;
import COM.myDomain.petPackage.petClass;
```

The following directs JServer Accelerator to compile all classes designated within this file: all classes in `myPackage`, `hisSubPackage` and the individual classes, `herClass` and `myClass`. These classes must have already been loaded into the database:

```
ncomp -user SCOTT/TIGER /tmp/jaccel/pubComped/pubworks.classes
```

Natively Compiling Packages That Are Fully or Partially Modified

If you change any of the classes within this JAR file, JServer Accelerator will only recompile shared libraries that contain the changed classes. It will not recompile all shared libraries designated in the JAR file. However, if you want all classes within a JAR file to be recompiled—regardless of whether they were previously natively compiled—you execute `ncomp` with the `-force` option, as follows:

```
ncomp -user scott/tiger -force pubProject.JAR
```

deploync

You can deploy any deployment JAR file with the `deploync` command. This includes the default output JAR file, <*file*>`_depl.jar` or the JAR created when you used the `ncomp -outputJarFile` option. The operating system and Oracle8*i* database version must be the same as the platform where it was natively compiled.

Note: The list of shared libraries deployed into JServer are listed within the JACCELERATOR$DLLS table.

Syntax

```
deploync [options] <deployment>.jar
  -user | -u <username>/<password>[@<database_url>]
  [-projectDir | -d <project_directory>]
  [-thin]
  [-oci8]
```

Argument Summary

Table 1–57 summarizes the `deploync` arguments.

Table 1–57 deploync Argument Summary

Argument	Description and Values
`<deployment>.jar`	The full pathname and filename of a deployment JAR file. This JAR file is created when you specify the -outputJarFile option on the ncomp tool. Note that deploync does not verify that this is a native compilation deployment JAR.
`-user | -u <username>/<password>[@<database>]`	Specifies a user, password, and database connect string; the files will be loaded into this database instance. The argument has the form *`<username>/<password>[@<database>]`*. If you specify the database URL on this option, you must specify it with OCI8 syntax. To provide a JDBC Thin database URL, use the -thin option.
`-projectDir | -d <absolute_path>`	Specifies the full path for the project directory. If not specified, JServer Accelerator uses the directory that ncomp is invoked from as the project directory.
`-thin`	The database URL that is provided on the -user option uses a JDBC Thin URL address for the database URL syntax.
`-oci8`	The database URL that is provided on the -user option uses an OCI8 URL address for the database URL syntax. However, if neither -oci8 or -thin are specified, the default assumes that you used an OCI8 database URL.

Example

Deploy the natively compiled deployment JAR file, `pub.jar`, to the `dbhost` database as follows:

```
deploync -user SCOTT/TIGER@dbhost:5521:orcl -thin /tmp/jaccel/PubComped/pub.jar
```

statusnc

After the native compilation is completed, you can check the status for your Java classes through the `statusnc` command. This tool will print out—either to the screen or to a designated file—the status of each class. In addition, the `statusnc` tool always saves the output within the JACCELERATOR$STATUS table. The values can be the following:

Class Native Compilation Status	Description
`ALREADY_NCOMPED`	The class is currently natively compiled.
`NEED_NCOMPING`	A class within the shared library was reloaded after native compilation. Thus, you should recompile this shared library.
`INVALID`	A class loaded in the database is invalid. JServer Accelerator tried to validate it and failed. The class will be excluded from the natively compiled shared library.

Note: The JACCELERATOR$STATUS table only contains the output from the last execution of the `statusnc` command. When executed, the `statusnc` command cleans out this table before writing the new records into it.

Syntax

```
statusnc [ options ] <class_designation_file>
   -user <user>/<password>[@database]
   [-output | -o <filename>]
   [-projectDir | -d <directory>]
   [-thin]
   [-oci8]
```

Argument Summary

Table 1–58 summarizes the `statusnc` arguments. The *`<class_designation_file>`* can be a *`<file>.jar`*, *`<file>.zip`*, or *`<file>.classes`*.

Table 1–58 statusnc Argument Summary

Argument	Description
`<file>.jar`	The full pathname and filename of a JAR file that was natively compiled.
`<file>.zip`	The full pathname and filename of a ZIP file that was natively compiled.
`<file>.classes`	The full pathname and filename of a classes file, which contains the list of classes that was natively compiled. See "Natively Compiling Specific Classes" on page 1-107 for a description of a classes file.

Table 1–58 statusnc Argument Summary

Argument	Description
`-user | -u <username>/<password> [@<database>]`	Specifies a user, password, and database connect string where the files are loaded. The argument has the form `<username>/<password>[@<database>]`. If you specify the database URL on this option, you must specify it with OCI8 syntax. To provide a JDBC Thin database URL, use the -thin option.
`-output <filename>`	Designates that the statusnc should output to the specified text file rather than to the screen.
`-projectDir | -d <absolute_path>`	Specifies the full path for the project directory. If not specified, JServer Accelerator uses the directory that ncomp is invoked from as the project directory.
`-thin`	The database URL that is provided on the -user option uses a JDBC Thin URL address for the database URL syntax.
`-oci8`	The database URL that is provided on the -user option uses an OCI8 URL address for the database URL syntax. However, if neither -oci8 or -thin are specified, the default assumes that you used an OCI8 database URL.

Example

```
statusnc -user SCOTT/TIGER -output pubStatus.txt /tmp/jaccel/PubComped/pub.jar
```

Miscellaneous Tools

This section describes special-purpose tools.

- java2rmi_iiop
- modifyprops

java2rmi_iiop

In the current JServer Enterprise JavaBeans implementation, EJBs communicate with clients by RMI-over-IIOP. This presents a difficulty for a CORBA client that wants to pass an object to an EJB for the EJB to invoke (call back) because the CORBA transport is IIOP, not RMI-over-IIOP. The CORBA client needs to pass the EJB an object the EJB can invoke with RMI-over-IIOP. The `java2rmi_iiop` tool generates the stubs, skeletons, and other classes a client or server needs to make an object remotely invocable by an EJB. (`java2rmi_iiop` is the analog of the VisiBroker for Java `java2iiop` tool, except that it expects interfaces that extend `java.rmi.Remote` rather than `org.omg.CORBA.Object`)

The Java interface definitions must follow the RMI spec:

- Interfaces must extend `java.rmi.Remote.`
- All remote methods must throw at least `java.rmi.RemoteException.`
- All arguments and return values of the remote methods must be valid RMI types.

Syntax

```
java2rmi_iiop [options] <file>.java ...
  [-no_bind]
  [-no_comments]
  [-no_examples]
  [-no_tie]
  [-root_dir <directory>]
  [-verbose]
  [-version]
  [-W <number>]
  [-wide]
```

Argument Summary

Table 1–59 summarizes the `java2rmi_iiop` arguments.

Table 1–59 java2rmi_iiop Argument Summary

Argument	Description
`-nobind`	Suppresses the generation of `bind()` methods.
`-no_comments`	Suppresses comments in generated code.
`-no_examples`	Suppresses the generation of example code.
`-no_tie`	Suppresses the generation of tie code.
`-root_dir`	Places all generated files in the specified directory instead of in the current directory.
`-verbose`	Emits extra messages.
`-version`	Displays the version of VisiBroker for Java that you are currently running.
`-W`	Setting this option to 0 (zero) suppresses all warnings from the compiler.
`-wide`	Maps Java `String`/`char` to IDL `wstring`/`wchar`.

Example

Generate RMI-over-IIOP class files for an RMI interface:

```
java2rmi_iiop Dictionary.java
```

modifyprops

Some aspects of the Oracle8*i* ORB are governed by properties it reads when a new session running the ORB starts. You can change these properties with the `modifyprops` tool. Developers should change ORB properties only when Oracle technical support provides instructions to do so.

Syntax

```
modifyprops {-u | -user} <user/password@<database> [options]
{<key> <value> [,<key> <value>] ... | <key> -delete}
  [-o | -oci8]
  [-t | -thin]
```

Argument Summary

Table 1–60 summarizes the `modifyprops` arguments.

Table 1–60 modifyprops Argument Summary

Argument	Description
`-user`	Specifies a user, password, and optional database connect string. See "user" on page 1-114 for details.
`-oci8`	Directs `modifyprops` to connect with the database using the OCI8 JDBC driver. `-oci8` and `-thin` are mutually exclusive; if neither is specified, then `-oci8` is used by default. Choosing `-oci8` implies the form of the database connect string. See "user" on page 1-114 for details.
`-thin`	Directs `modifyprops` to communicate with the database using the thin JDBC driver. `-oci8` and `-thin` are mutually exclusive; if neither is specified, then `-oci8` is used by default. Choosing `-thin` implies the form of database connect string See "user" on page 1-114 for details.
`<key> <value>`	Oracle technical support will advise you of the values to enter for `<key>` and `<value>`.

Argument Details

user

`{-user | -u}` <user>/<password>[@<database>]

The permissible forms of `@<database>` depend on whether you specify `-oci8` or `-thin`; `-oci8` is the default.

- `-oci8`: `@<database>` is optional. If you do not specify, then `modifyprops` uses the user's default database. If specified, then `<database>` can be a TNS name or a Net8 name-value list.
- `-thin`: `@<database>` is required. The format is `<host>:<lport>:<SID>`.
  - `<host>` is the name of the machine running the database.
  - `<lport>` is the listener port that has been configured to listen for Net8 connections. In a default installation, it is 5521.
  - `<SID>` is the database instance identifier. In a default installation it is `ORCL`.

2 Backwards Compatibility Tools

The underlying logic for the session shell, publish, and remove tools were changed for Release 8.1.7. Because of this, each of these tools are not backward compatible to versions of Oracle8*i* Release 8.1.6 and prior. Thus, the following tools are provided for backwards compatibility: `sess_sh_816`, `publish_816`, and `remove_816`. These tools will be deprecated in Release 8.2.

The tools described in this chapter as follows:

- Session Namespace Tools
- publish_816
- remove_816
- sess_sh_816

Session Namespace Tools

Each database instance running the Oracle8*i* JServer software has a session namespace, which the Oracle8*i* ORB uses to activate CORBA and EJB objects. A *session namespace* is a hierarchical collection of objects known as PublishedObjects and PublishingContexts. PublishedObjects are the leaves of the hierarchy and PublishingContexts are the nodes, analogous to UNIX file system files and directories. Each PublishedObject is associated with a class schema object that represents a CORBA or EJB implementation. To activate a CORBA or EJB object, a client refers to a PublishedObject's name. From the PublishedObject, the Oracle8*i* ORB obtains the information necessary to find and launch the corresponding class schema object.

Creating a PublishedObject is known as *publishing* and can be done with the command-line `publish_816`tool or the interactive session shell, both of which this section describes. CORBA server developers create PublishedObjects explicitly after loading the implementation of an object with `loadjava`. EJB developers do not explicitly load or publish their implementations; the `deployejb` tool implicitly does both.

A *PublishedObject* has the following *attributes*:

- Schema Object Name: the name of the Java class schema object associated with the PublishedObject.
- Schema: the name of the schema containing the corresponding class schema object.
- Helper Schema Object Name: the name of the helper class the Oracle8*i* ORB uses to automatically narrow a reference to an instance of the CORBA object or EJB.

PublishedObjects and PublishingContexts, as with their file and directory counterparts, have owners and rights (privileges). An owner can be a user name or a role name; only the owner can change the ownership or rights of a PublishedObject or PublishingContext. Table 2–1 describes session namespace rights.

Table 2–1 PublishingContext and PublishedObject Rights

Right	Meaning for PublishingContext	Meaning for PublishedObject
read	List contents and attributes (type, rights and creation time).	List object attributes (type, schema object, schema, helper, rights, and creation time).

Table 2–1 PublishingContext and PublishedObject Rights (Cont.)

Right	Meaning for PublishingContext	Meaning for PublishedObject
write	Create a PublishedObject or PublishingContext in the PublishingContext.	Republish object.
execute	Use contents to resolve a name.	Activate associated class.

Oracle8*i* creates a session namespace automatically when the Oracle8*i* ORB is configured. The PublishingContexts contained in Table 2–2 are present in all session namespaces:

Table 2–2 Initial PublishingContexts and Rights

Name	Owner	Read	Write	Execute
`/`	`SYS`	`PUBLIC`	`SYS`	`PUBLIC`
`/bin`	`SYS`	`PUBLIC`	`SYS`	`PUBLIC`
`/etc`	`SYS`	`PUBLIC`	`SYS`	`PUBLIC`
`/test`	`SYS`	`PUBLIC`	`PUBLIC`	`PUBLIC`

Because by default only `/test` is writable by PUBLIC, you will normally create PublishingContexts and PublishedObjects subordinate to `/test`.

publish_816

The `publish_816` tool creates or replaces (republishes) a PublishedObject in a PublishingContext. It is not necessary to republish when you update a Java class schema object; republishing is required only to change a PublishedObject's attributes. To publish, you must have write permission (the write right) for the destination PublishingContext; by default only the PublishingContext `/test` is writable by PUBLIC. To republish you must additionally have the write right for the PublishedObject.

Syntax

```
publish_816 [options]
<name> <class> [<helper>] -user <username> -password <password>
-service <serviceURL>
```

where options are:

```
  [-describe]
  [{-g | -grant} {<user> | <role>}[,{<user> | <role>}]...]
  [{-h | -help}]
  [-idl]
  [-iiop]
  [-replaceIDL]
  [-role <role>]
  [-republish]
  [-schema <schema>]
  [-keepcase]
  [-ssl]
  [-useServiceName]
  [-version]
```

Argument Summary

Table 2–3 summarizes the `publish_816` tool arguments.

Table 2–3 publish_816 Tool Argument Summary

Option	Description
`<name>`	Name of the PublishedObject being created or republished; PublishingContexts are created if necessary.
`<class>`	Name of the class schema object that corresponds to <name>.
`<helper>`	Name of the Java class schema object that implements the `narrow()` method for `<class>`.
`-user`	Specifies identity with which to log into the database instance named in `-service`.
`-password`	Specifies authenticating password for the username specified with `-user`.

Table 2–3 publish_816 Tool Argument Summary (Cont.)

Option	Description
`-service`	URL identifying database whose session namespace is to be "opened" by `sess_sh_816`. The serviceURL has the form: `sess_iiop://<host>:<lport>:<sid>`. `<host>` is the computer that hosts the target database; `<lport>` is the listener port that has been configured to listen for session IIOP; `<sid>` is the database instance identifier. Example: `sess_iiop://localhost:2481:orcl` which matches the default installation on the invoker's machine.
`-describe`	Summarizes the tool's operation, then exits.
`-grant`	After creating or republishing the PublishedObject, grants read and execute rights to the sequence of `<user>` and `<role>` names. When republishing, replace the existing users / roles that have read / execute rights with the `<user>` and `<role>` names. To selectively change the rights of a PublishedObject, use the `sess_sh_816`'s `chmod` command. Note that to activate a CORBA object or EJB, a user must have the execute right for both the PublishedObject and the corresponding class schema object. The sequence of user and role names must be a comma-separated list, containing no internal spaces.
`-help`	Summarizes the tool's syntax, then exits.
`-idl`	Load the IDL interface definition into the IFR.
`-iiop`	Connects to the target database with IIOP instead of the default session IIOP. Use this option when publishing to a database server that has been configured without session IIOP.
`-replaceIDL`	If an IDL interface definition currently exists within the IFR, replace it with this version. If not specified, the publish command will not replace the existing interface within the IFR. The -replaceIDL flag will replace any interface with the same name in the IFR, even if it was originally stored by another user. Thus, different users can overwrite another user's interface unknowingly.
`-role`	Role to assume for the publish; no default.

Table 2–3 publish_816 Tool Argument Summary (Cont.)

Option	Description
`-republish`	Directs `publish_816` to replace an existing PublishedObject; without this option, the `publish_816` tool rejects an attempt to publish an existing name. If the PublishedObject does not exist, `publish_816` creates it. Republishing deletes non-owner rights; use the `-grant` option to add read/execute rights when republishing.
`-schema`	The schema containing the Java <class> schema object. If you do not specify, the `publish_816` tool uses the invoker's schema.
`-keepcase`	Normally, any schema name supplied is uppercased by default. If you created a schema name that requires lowercase letters, specify the -keepcase option. Thus, you would execute publish_816 ... -schema mySchema -keepcase ...
`-ssl`	Connects to the database with SSL server authentication. You must have configured the database for SSL to use this option, and you must specify an SSL listener port in `-service`.
`-useServiceName`	If you are using a service name instead of an SID in the URL, you must specify this flag. Otherwise, the tool assumes the last string in the URL is the SID.
`-version`	Shows the tool's version, then exist.

Here is a `publish_816` example.

Publish the CORBA server implementation `vbjBankTestbank.AccountManagerImpl` and its helper class as `/test/bankMgr` in the tool invoker's schema:

```
publish_816 /test/bankMgr vbjBankTestServer.AccountManagerImpl \
vbjBankTestServer.AccountManagerHelper \
-user SCOTT -password TIGER \
-service sess_iiop://dlsun164:2481:orcl
```

remove_816

The `remove_816` tool removes a PublishedObject or PublishingContext from a session namespace. It does not `remove_816` the Java class schema object associated with a PublishedObject; use `dropjava` to do that.

Syntax

```
remove_816 <name> -user <username> -password <password> -service <serviceURL>
[options]
  [{-d | -describe}]
  [{-h | -help}]
  [-iiop]
  [{-r | -recurse}]
  [-role role]
  [-ssl]
  [-useServiceName]
  [-version]
```

Argument Summary

Table 2–4 describes the `remove_816` arguments.

Table 2–4 remove_816 Argument Summary

Option	Description
`<name>`	Name of PublishingContext or PublishedObject to be removed.
`-user`	Specifies identity with which to log into the instance named in `-service`.
`-password`	Specifies authenticating password for the `<username>` you specified with `-user`.
`-service`	URL identifying database whose session namespace is to be "opened" by `sess_sh_816`. The serviceURL has the form: `sess_iiop://`*`<host>`*`:`*`<lport>`*`:`*`<sid>`*. `<host>` is the computer that hosts the target database; `<lport>` is the listener port that has been configured to listen for session IIOP; `<sid>` is the database instance identifier. Example: `sess_iiop://localhost:2481:orcl` which matches the default installation on the invoker's machine.
`-describe`	Summarizes the tool's operation, then exits.
`-help`	Summarizes the tool's syntax, then exits.
`-iiop`	Connects to the target database with IIOP instead of the default session IIOP. Use this option when removing from a database server that has been configured without session IIOP.

Table 2–4 remove_816 Argument Summary (Cont.)

Option	Description
`-recurse`	Recursively removes `<name>` and all subordinate PublishingContexts; required to remove a PublishingContext.
`-role`	Role to assume for the remove; no default.
`-ssl`	Connects to the database with SSL server authentication. You must have configured the database for SSL to use this option.
`-useServiceName`	If you are using a service name instead of an SID in the URL, you must specify this flag. Otherwise, the tool assumes the last string in the URL is the SID.
`-version`	Shows the tool's version, then exits.

Here are examples of `remove_816` tool usage.

- Remove a PublishedObject named `/test/testhello`:

```
remove_816 /test/testhello -user SCOTT -password TIGER \
-service sess_iiop://dlsun164:2481:orcl
```

- Remove a PublishingContext named `/test/etrader`:

```
remove_816 -r /test/etrader -user SCOTT -password TIGER \
-service sess_iiop://dlsun164:2481:orcl
```

sess_sh_816

The `sess_sh_816` (session shell) tool is an interactive interface to a database instance's session namespace. You specify database connection arguments when you start `sess_sh_816`. It then presents you with a prompt to indicate that it is ready for commands.

The `sess_sh_816` gives a session namespace much of the "look and feel" of a UNIX file system you access through a shell, such as the C shell. For example, the session shell command:

```
ls /alpha/beta/gamma
```

means "List the PublishedObjects and PublishingContexts in the PublishingContext known as `/alpha/beta/gamma`". (NT users note: `/alpha/beta/gamma`, not `\alpha\beta\gamma`.) Indeed, many session shell command names that operate on PublishingContexts have the same names as their UNIX shell counterparts that operate on directories. For example: mkdir (create a PublishingContext) and cd (change the working PublishingContext).

In addition to UNIX-style manipulation of PublishingContexts and PublishedObjects, the session shell can launch an *executable*, which is analogous to a Java standalone application, that is, a class with a static main() method. Executables must have been loaded with loadjava, but not published—publishing is for CORBA and EJB objects only.

Syntax

```
sess_sh_816 [options] -user <user> -password <password> -service <serviceURL>
  [-d | -describe]
  [-h | -help]
  [-iiop]
  [-role <rolename>]
  [-ssl]
  [-useServiceName]
  [-version]
```

Argument Summary

Table 2–5 summarizes the sess_sh_816 command line arguments.

Table 2–5 sess_sh_816 Argument Summary

Option	Description
-user	Specifies user's name for connecting to the database.
-password	Specifies user's password for connecting to the database.
-service	URL identifying database whose session namespace is to be "opened" by sess_sh_816. The serviceURL has the form: `sess_iiop://<host>:<lport>:<sid>`. <host> is the computer that hosts the target database; <lport> is the listener port configured to listen for session IIOP; <sid> is the database instance identifier. Example: `sess_iiop://localhost:2481:orcl` which matches the default database installation on the invoker's machine.
-describe	Summarizes the tool's operation, then exits.
-help	Summarizes the tool's syntax, then exits.

Table 2–5 sess_sh_816 Argument Summary (Cont.)

Option	Description
`-iiop`	Connects to the target database with plain IIOP instead of the default session IIOP. Use this option for a database server configured without session IIOP.
`-role`	Role to pass to database; there is no default.
`-ssl`	Connect to the database with SSL server authentication. You must have configured the database for SSL and specify an SSL port to use this option.
`-useServiceName`	If you are using a service name instead of an SID in the URL, you must specify this flag. Otherwise, the tool assumes the last string in the URL is the SID.
`-version`	Shows the command's version, then exits.

Here is a `sess_sh_816` example.

Open a session shell on the session namespace of the database `orcl` on listener port 2481 on host `dbserver`.

```
sess_sh_816 -user scott -password tiger -service sess_iiop://dbserver:2481:orcl
```

cd Command

The `cd` command is analogous to a UNIX shell's `cd` command; it changes the working PublishingContext.

Syntax

```
cd [path]
```

Here is an example.

Change to root PublishingContext:

```
$ cd /
```

chmod Command

The `chmod` command is analogous to a UNIX shell's `chmod` command; it changes the users or roles that have rights for a PublishingContext or PublishedObject. See Table 2–1 on page 2-2 for descriptions of the read, write, and execute rights. Only the object's owner can change its rights.

Syntax

```
chmod  [options] {+|-}{r|w|e} {<user> | <role>} [, {<user> | <role>} ...] \
<objectname>
  [-h | -help]
  [-version]
```

Argument Summary

Table 2–6 summarizes the `chmod` arguments.

Table 2–6 chmod Argument Summary

Option	Description
`+/-rwe`	Specifies the right (read, write, or execute) to be added (+) or removed (-) for `<user>` or `<role>`.
`<user> | <role>`	Specifies the user or role whose rights are to be increased or decreased.
`<objectname>`	Specifies the name of the PublishingContext or PublishedObject whose rights are to be changed.
`-help`	Summarizes the command's syntax, then exits.
`-version`	Shows the command's version, then exits.

Here are some chmod examples.

- Give execute rights for `/alpha/beta/gamma` to Scott and Nancy:

  ```
  $ chmod +x scott nancy /alpha/beta/gamma
  ```

- Remove Scott's write rights for the same object:

  ```
  $ chmod -w scott /alpha/beta/gamma
  ```

chown Command

The chown command is analogous to the UNIX chown command; it changes the ownership of a PublishingContext or PublishedObject. The owner of a newly created PublishingContext or PublishedObject is the user who publishes it. To change a PublishingContext's or PublishedObject's ownership you must be SYS.

Syntax

```
chown [options] {<user> | <role>} <objectname>
  [-h | -help]
```

```
  [-version]
```

Argument Summary

Table 2–7 summarizes the chown arguments.

Table 2–7 chown Argument Summary

Option	Description
`<user> | <role>`	Specifies the user or role to be the new owner.
`<objectname>`	Specifies the name of the PublishingContext or PublishedObject whose owner is to be changed.
`-help`	Summarizes the command's syntax, then exits.
`-version`	Shows the command's version, then exits.

Here is a chown example.

Make Scott the owner of /alpha/beta/gamma:

```
$ chown scott /alpha/beta/gamma
```

exit Command

The exit command terminates sess_sh_816.

Syntax

```
exit
```

Here is an example:

Leave the session shell:

```
$ exit
%
```

help Command

The help command summarizes the syntax of the session shell commands.

Syntax

```
help
```

Here is a `help` example.

```
$ help
Commands are of the format <command> [arg1, ar2...]
Intrinsic Commands:
    exit           exit the shell
    help           prints this message
    version        print version inforamtion
    pwd            print working directory
    cd             change working directory
    ls             list directory
    ln             link name
    chmod          change read, write or execute permissions on an object
    chown          change an objects owner
    mkdir          create a directory
    mv             move an object or directory to another location
    rm             remove an object or directory
    lpwd           print local file system working directory
    publish        publish an object
    republish      republish an object
    java           execute the "main" method on a java class
```

java Command

The `java` command is analogous to the JDK `java` command; it invokes a class's static `main()` method. The class must have been loaded with `loadjava`. (There is no point to publishing a class that will be invoked with the `java` command.) The `java` command provides a convenient way to test Java code that runs in the database. In particular, the command catches exceptions and redirects the class's standard output and standard error to the session shell, which displays them as with any other command output. (The usual destination of standard out and standard error for Java classes executed in the database is one or more database server process trace files, which are inconvenient and may require DBA priviliges to read.)

Syntax

```
java class [-schema <schema>] [arg1 ... argn] [options]
  [{-h | -help}]
  [-version]
```

Argument Summary

Table 2–8 summarizes the `java` arguments.

Table 2–8 java Argument Summary

Option	Description
`class`	Names the Java class schema object that is to be executed.
`-schema`	Names the schema containing the class to be executed; the default is the invoker's schema.
`arg1 ... argn`	Arguments to the class's `main()` method.
`-help`	Summarizes the command's syntax, then exits.
`-version`	Shows the command's version, then exits.

Here is a `java` command example.

Say hello and display arguments:

```
package hello;
public class World {
    public World() {
        super();
    }
    public static void main(String[] argv) {
        System.out.println("Hello from the JServer/ORB");
        if (argv.length != 0)
            System.out.println("You supplied " + argv.length + " arguments: ");
            for (int i = 0; i < argv.length; i++)
                System.out.println(" arg[" + i + "] : " + argv[i]);
    }
}
```

Compile, load, publish, and run the executable as follows, substituting your userid, host, and port information as appropriate:

```
% javac hello/World.java
% loadjava -r -user scott/tiger@localhost:2481:orcl hello/World.class
% sess_sh_816 -user scott -password tiger -service
sess_iiop://localhost:2481:orcl
$ java testhello alpha beta
Hello from the JServer/ORB
You supplied 2 arguments:
arg[0] : alpha
arg[1] : beta
$
```

ln Command

The `ln` (link) command is analogous to the UNIX `ln` command. A link is a synonym for a PublishingContext or PublishedObject. A link can prevent a reference to a PublishingContext or PublishedObject from becoming invalid when you move a PublishingContext or PublishedObject (see "mv Command" on page 2-18); creating a link with the old name makes the object accessible by both its old and new names.

Syntax

```
ln <object> <link>
```

Argument Summary

Table 2–9 summarizes the `ln` arguments.

Table 2–9 ln Argument Summary

Option	Description
`<object>`	The name of the PublishingContext or PublishedObject for which a link is to be created.
`<link>`	The synonym by which `<object>` is also to be known.

Here is an `ln` command example.

Preserve access through `old`, although the object's name is changed to `new`:

```
$ mv old new
$ ln new old
```

lpwd Command

The `lpwd` (local print working directory) command displays the name of the working directory, just as executing `pwd` outside of the session shell would.

Syntax

```
lpwd
```

Here is an example of the `lpwd` command that shows the working directory:

```
$ lpwd
/home/usr/billc
```

ls Command

The `ls` (list) command shows the contents of PublishingContexts as the UNIX `ls` command shows the contents of directories.

Syntax

```
ls [options] [{<pubcon> | <pubobj} [{<pubcon> | <pubobj}] ...]
  [-dir]
  [-h | -help]
  [-l]
  [-ld | ldir]
  [-R]
  [-version]
```

Argument Summary

Table 2–10 describes the `ls` arguments.

Table 2–10 ls Argument Summary

Option	Description
`<pubcon> | <pubobj>`	Name of PublishingContext(s) and / or PublishingObject(s) to be listed; the default is the working PublishingContext.
`-dir`	Shows only PublishingContexts; analogous to the UNIX `ls -d` command.
`-help`	Summarizes the command’s syntax, then exits.
`-l`	Shows contents in long (detailed) format. The long format includes name, creation time, owner, and rights. For PublishedObjects, the option also shows class, schema, and helper.
`-ldir`	Lists PublishingContexts in long format, ignoring PublishingObjects; analogous to UNIX `ls -ld` command.
`-R`	Lists recursively.
`-version`	Shows the command’s version, then exits.

Here are examples of the `ls` command.

Show contents of the root PublishingContext in short format:

```
$ ls /
bin/
etc/
```

```
test/
```

Show contents of the root PublishingContext in long format:

```
$ ls -l /
Read    Write   Exec      Owner    Date   Time    Name      Schema    Class     Helper
PUBLIC  SYS     PUBLIC    SYS      Dec 14 14:59  bin/
PUBLIC  SYS     PUBLIC    SYS      Dec 14 14:59  etc/
PUBLIC  PUBLIC  PUBLIC    SYS      Dec 14 14:59  test/
```

Show contents of the `/test` PublishingContext in long format:

```
$ ls -l test
Read  Write Exec  Owner Date   Time  Name Schema Class                  Helper
SCOTT SCOTT SCOTT SCOTT Dec 14 16:32 bank SCOTT  Bank.AccountManagerImpl Bank.AccountManagerHelper
```

mkdir Command

The `mkdir` command is analogous to the UNIX shell `mkdir` command; it creates a PublishingContext. You must have the write right for the target PublishingContext to use `mkdir` in it.

Syntax

```
mkdir [options] <name>
 [-path]
```

Argument Summary

Table 2–11 describes the `mkdir` arguments.

Table 2–11 mkdir Argument Summary

Option	Description
`<name>`	Name of PublishingContext to create.
`-path`	Creates intermediate PublishingContexts if they do not exist.

Here are examples of the `mkdir` command.

Create a PublishingContext called `/test/alpha` (`/test` exists):

```
mkdir /test/alpha
```

Create a PublishingContext called `/test/alpha/beta/gamma` (`/test/alpha/beta` does not exist):

```
$ mkdir -path /test/alpha/beta/gamma
```

mv Command

The `mv` command is analogous to the UNIX shell `mv` command.

Syntax

```
mv <old> <new>
```

Here is an example of the `mv` command.

Change the name of `/test/foo` to `/test/bar`:

```
$ mv /test/foo /test/bar
```

publish Command

The `publish` command creates or replaces (republishes) a PublishedObject in a PublishingContext. It is not necessary to republish when you update a Java class schema object that has been published; republish only to change a PublishedObject's attributes. To publish, you must have the write right for the destination PublishingContext; to republish you must also have the write right for the PublishedObject.

Syntax

```
publish <name> <class> <helper> [options]
  [{-e | -executable}]
  [{-g | -grant} {<user> | <role>}[,{<user> | <role>} ... ]]
  [{-h | -help}]
  [-republish]
  [-schema <schema>]
  [-version]
```

Argument Summary

Table 2–12 summarizes the `publish` command arguments.

Table 2–12 publish Command Argument Summary

Option	Description
`<name>`	Name of the PublishedObject being created or republished; PublishingContexts are created if necessary.
`<class>`	Name of the class schema object that corresponds to <name>.
`<helper>`	Name of the Java class schema object that implements the `narrow()` method for `<class>`.
`-grant`	After creating or republishing the PublishedObject, grants read and execute rights to the sequence of `<user>` and `<role>` names. When republishing, replaces the existing users/roles that have read/execute rights with the `<user>` and `<role>` names. To selectively change the rights of a PublishedObject, use the `session shell`'s `chmod` command. Note that to activate a CORBA object or EJB, a user must have the execute right for both the PublishedObject and the corresponding class schema object.
`-help`	Summarizes the command's syntax, then exits.
`-republish`	Directs `publish` to replace an existing PublishedObject; without this option, the `publish` command rejects an attempt to publish an existing name. If the PublishedObject does not exist, it is created. Republishing deletes non-owner rights; use the `-grant` option to add read/execute rights when republishing.
`-schema`	The schema containing the Java `<class>` schema object; if you do not specify, the command uses the invoker's schema.
`-version`	Shows the command's version, then exits.

Here is an example of the `publish` command.

Publish the CORBA server implementation `Bank.AccountManagerImpl` and its helper class as `/test/bank` in the command invoker's schema:

```
$ ls -l /test
$ publish /test/bank Bank.AccountManagerImpl Bank.AccountManagerHelper
$ ls -l /test
Read  Write Exec  Owner Date   Time  Name  Schema Class                   Helper
SCOTT SCOTT SCOTT SCOTT Dec 14 16:32 bank  SCOTT  Bank.AccountManagerImpl Bank.AccountManagerHelper
```

pwd Command

The pwd command displays the name of the current working PublishingContext. It is analogous to the UNIX `pwd` command.

Syntax

```
pwd
```

Here is an example of the `pwd` command.

```
$ pwd
/test/alpha
```

rm Command

The `rm` (remove) command is analogous to the `rm -r` UNIX shell commands; it removes a PublishedObject or a PublishingContext, including its contents. To remove an object, you must have the write right for the containing PublishingContext.

Syntax

```
rm [options] <object> ... <object>
  [{-h | -help}]
  [-r]
  [-version]
```

Argument Summary

Table 2–13 describes the `rm` arguments.

Table 2–13 rm Argument Summary

Option	Description
`<object>`	Name of PublishedObject or PublishingContext to be removed.
`-help`	Summarizes the command's syntax, then exits.
`-r`	Interprets `<object>` as a PublishingContext; removes it and its contents recursively.
`-version`	Shows the command's version, then exits.

Here is an example of the `rm` command.

Remove the PublishedObject `/test/bank`:

```
rm /test/bank
```

Remove the PublishingContext `/test/release3` and everything it contains:

```
rm -r /test/release3
```

version Command

The `version` command shows the version of the `sess_sh_816` tool.

Syntax

```
version
```

Here is an example of the `version` command.

Display the session shell's version:

```
$ version
1.0
```

Index